Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана ПУЛЮЯ

кафедра програмної інженерії

Навчальний посібник

Лабораторний практикум з розділу «Шаблони проектування» дисципліни «Архітектура та проектування програмного забезпечення»

Тернопіль 2016

Петрик М., Михалик Д., Мудрик І., Стоянов Ю. Лабораторний практикум з розділу «Шаблони проектування» дисципліни «Архітектура та проектування програмного забезпечення: навчальний посібник, Тернопіль: ТНТУ імені Івана Пулюя, 2016. 36 с.

Навчальний посібник містить практичні рекомендації для допомоги студентам у вивченні дисципліни “Архітектура та проектування програмного забезпечення”, зокрема передбачають дослідження моделей архітектур, патернів та методологій розробки програмного забезпечення, використаня методів практико-орієнтованого навчання, зокрема: процес аналізу предметної області; розробки моделі варіантів використання із застосуванням шаблонів проектування; розробки діаграми класів; опису класів та їх атрибутів, методів; опису зв’язків та залежностей між класами; програмну реалізацію у вигляді C++ коду; розробки методів і сценаріїв способу застосування таких реалізацій. Розроблено з використанням ліцензованого інструментального середовища проектування IBM Rational Software Architect. Посібник орієнтований для студентів спеціальності 121 інженерія програмного забезпечення для набуття практичних навиків застосування архітектурних моделей та патернів розробки програмного забезпечення.

Укладачі: М. Петрик, Д. Михалик, І. Мудрик, Ю. Стоянов
Відповідальний за випуск: М. Петрик
Рецензент: С. Лупенко

Розглянуто на засіданні кафедри програмної інженерії, протокол №1 від 16.08.2016р.

Схвалено на засіданні методичної ради факультету комп’ютерно-інформаційних систем і програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя, протокол №1 від 1 вересня 2016 р.

ЗМІСТ

Вступ

Лабораторні роботи передбачають дослідження моделей архітектур та методологій розробки програмного забезпечення. Для виконання завдань лабораторних робіт слід ознайомитися з теоретичним матеріалом, провести дослідження завдання відповідно до варіанту і результати дослідження відобразити у звіті до лабораторної роботи. Варіант завдання обираємо відповідно до номера студента у списку підгрупи.

Звіт до лабораторних робіт слід у вказані терміни роздрукувати, скріпити та представити до захисту, електронний варіант звіту завантажити в скриньку завдань за номером лабораторної роботи дистанційного курсу.

Результатом виконання кожної лабораторної роботи є оформлений звіт, з яким студент повинен захистити роботу.

Узагальнена структура звіту:

1. Титульний аркуш, який повинен включати наступні елементи:

- назву ВНЗ: Тернопільський національний технічний університет імені Івана Пулюя;
- назву факультету: Факультет комп'ютерно-інформаційних систем та програмної інженерії;
- назву кафедри;
- назву дисципліни, порядковий номер та тему лабораторної роботи, до якої оформляється звіт;
- інформацію про виконавця: номер групи, прізвище ініціали;
- рік захисту роботи.

2. Основної частина звіту, яка повинна включати:

- мету роботи;
- завдання, винесене на роботу та номер варіанту;
- опис предметної області;
- модель варіантів використання (в IBM RSA);
- діаграму класів (в рамках даної моделі, лише необхідний мінімум класів);
- описи класів;
- програмна реалізація: Компонентна модель (опис структури коду);
- код автоматично згенерований по діаграмі класів в IBM RSA, мова C++ або Java;
- код розбитий на модулі/ компоненти;
- в оформленні коду дотримуватись єдиного стилю та доповнювати код коментарями;
- основні методи і сценарії показати в діаграмі послідовності;
- отримані результати дослідження чи результати виконання кожного з пункту завдання лабораторної роботи (діаграми, лістинг коду, результати тестування і ін.).

3. Висновки, які повинні включати відповіді на поставлені мету та питання в завданні лабораторної роботи,

Лабораторні виконуються в програмному середовищі IBM Rational Software Architect, http://www.ibm.com/developerworks/downloads/r/architect/index.html [1].

Генерація коду з UML-діаграми проводиться згідно інструкцій на сайті [2,3,4].

При оформленні звіту рекомендовано використовувати довільний текстовий редактор, звіт оформляти на аркуші формату А4 (відступи В,Н - 20 мм, П-15 мм, Л-25 мм.). Для звичайного тексу використовувати шрифт Times New Roman 14 пт, вирівняний по ширині сторінки, міжрядковий інтервал - 1,5. Для лістингу коду використовувати шрифт Courier 10 пт, вирівняний по лівому краю, міжрядковий інтервал - 1. Усі лістинги коду та рисунки повинні мати номери та підписи до них з посиланням на них з тексту звіту.

Оцінювання:

- 2 тестові модулі по 15 та 20 балів відповідно;
- 8 лабораторних робіт на 40 балів сумарно (5 балів за кожну лабораторну роботу);
- 25 балів екзамен (для груп СП), або 1/3 балів залік (для решти груп).

Лабораторна робота 1.
Дослідження архітектурних моделей та методологій розробки ПЗ для обраної предметної області. Ознайомлення з базовими елементами проектування та реалізації програмних систем в середовищі проектування IBM RSA

Тема: Дослідження архітектурних моделей та методологій розробки програмного забезпечення для обраної предметної області. Ознайомлення з базовими елементами проектування та реалізації програмних систем в середовищі проектування RSA IBM.

Мета роботи*:*

• дослідити базові моделі архітектур програмного забезпечення;

• дослідити методології та підходи до розробки ПЗ;

• ознайомитися з принципом ООП - SOLID;

• обрати базову для реалізації архітектуру обраної предметної області. Описати підхід (з точки методології розробки ПЗ) до розробки програмного продукту згідно обраної предметної області. Визначити переваги та недоліки застосування моделі архітектури;

• ознайомитися з середовищем проектування IBM Rational Software Architect.

• проаналізувати обрану предметну область. Визначити вимоги, акторів та базові прецеденти програмного продукту. Побудувати діаграму ВВ (прецедентів) та діаграму Класів, використовуючи можливості середовища проектування IBM RSA.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з основними архітектурами реалізації ПЗ. Ознайомтесь з методологіями розробки ПЗ. Розгляньте теоретичний матеріал щодо принципів побудови Хорошої архітектури SOLID.

2. Оберіть тематику власної розробки. Основні принципи моделі архітектури. Виділіть основні функції проектованої інформаційної системи та зовнішні сутності, з якими система взаємодіє. Тематику можна обрати власну, або згідно списків №1,2,3 на стр. 8.

3. Ознайомтесь з середовищем проектування IBM Rational Software Architect. Побудуйте в даному середовищі проектування діаграми прецедентів (ВВ), класів обраної Вами предметної області.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості

1. **Архітектура програмного забезпечення** (англ. software architecture) - це структура програми або обчислювальної системи, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. Цей термін стосується також документування архітектури програмного забезпечення. Документування архітектури ПЗ спрощує процес комунікації між зацікавленими особами (англ. stakeholders), дозволяє зафіксувати прийняті на ранніх етапах проектування рішення про високорівневий дизайн системи і дозволяє використовувати компоненти цього дизайну і шаблони повторно в інших проектах.

Архітектура - це принцип організації компонентів усередині системи: їх кількість, якість, інтерфейси і протоколи взаємодії. Що залежить від архітектури? Від неї залежить ціна та витрати на підтримку і розробку нових фіч, трудовитрати на побудову цілої системи з використанням даної архітектури. Тобто формально від архітектури залежить найважливіший параметр розробки - собівартість. А побічно ще і можливість повторного використання коду, а разом з ним і зменшення трудовитрат на кожну подальшу розробку.
Вибір або створення архітектури залежить від конкретних завдань. Наприклад, наскільки універсальним планується додаток, які модулі повинні бути присутніми, яка запланована навантаження на ресурс.

2. **Вибір архітектури ПЗ** — це етап проектування, що виконується після етапу аналізу і формулювання вимог. Задача такого проектування — перетворення вимог до системи у вимоги до ПЗ і побудова на їх основі архітектури системи. Побудова архітектури системи здійснюється шляхом визначення цілей системи, її вхідних і вихідних даних, декомпозиції системи на підсистеми, компоненти або модулі та розроблення її загальної структури. Проектування архітектури системи може проводитися різними методами (стандартизованим, об'єктно-орієнтованим, компонентним і ін.), кожний з яких пропонує свій шлях побудови архітектури, а саме, визначення концептуальної, об'єктної й інших моделей за допомогою відповідних конструктивних елементів (блок-схем, графів, структурних діаграм тощо).
3. **Методологія** - це система принципів, а також сукупність ідей, понять, методів, способів і засобів, які визначають стиль розробки програмного забезпечення. Це - реалізація стандарту. Самі стандарти лише вказують на те, що повинно бути, залишаючи свободу вибору і адаптації. Методології являють собою ядро теорії управління розробкою програмного забезпечення. До існуючої класифікації, залежно від використовуваної моделі життєвого циклу (каскадні і ітераційні методології) додалася більш загальна класифікація на прогнозовані і адаптивні методології.

Базовими підходами вважають:

- Rational Unified Process (RUP).
- Microsoft Solutions Framework (MSF).
- Методологія аналізу вимог і гнучкий процес розробки ПЗ ICONIX, що ґрунтується на варіантах використання (пропонований фірмою ICONIX Software Engineering, Inc.).
- Швидка розробка додатків — Rapid Application Development (RAD).
- Метод розробки динамічних систем — Dynamic Systems Development Method (DSDM).

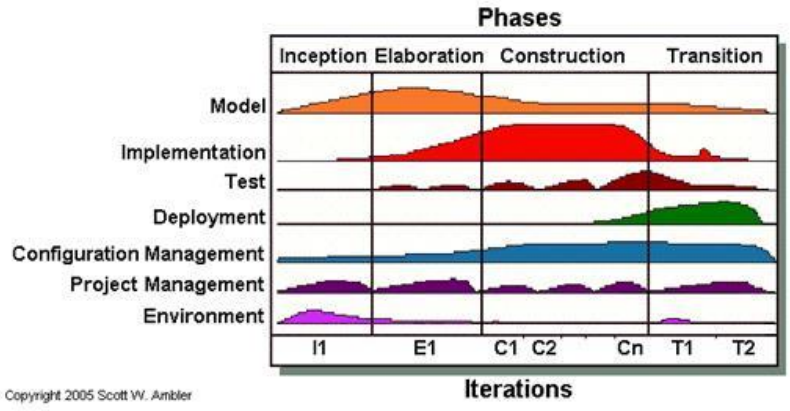

Рис. 1.1. Фази RUP – розробки ПЗ

4. Принципи "хорошої" архітектури SOLID

Рис. 1.2. Принципи SOLID

Принцип Єдиної Відповідальності (Single Responsibility Principle)

Клас повинен мати тільки одну причину для зміни

Принцип відкриття-закриття (Open Close Principle або OCP)

Програмні сутності такі як класи, модулі та функції повинні бути відкриті для розширення, але закриті для змін.

Принцип Заміщення Ліскоу (Liskov's Substitution Principle)

Похідні типи повинні бути здатні повністю заміщатися їх базовими типами.

Принцип Відділення Інтерфейсу (Interface Segregation Principle)

Клієнти не повинні бути залежними від інтерфейсів, які вони не використовують. Інтерфейси містять методи, які не є специфічними для них, такі методи призводять до того, що інтерфейси називають забрудненими або жирними. Ми повинні уникати створення таких інтерфейсів.

Принцип інверсії залежностей

(Dependency Inversion Principle) - залежності всередині системи будуються на основі абстракцій. Модулі верхнього рівня не залежать від модулів нижнього рівня. Абстракції не залежать від подробиць.

Список 1. Варіанти архітектурних моделей:

1. Архітектура класної дошки (Blackboard)
2. 2-х рівнева клієнт-серверна архітектура (2-tier Client-server)
3. 3-х рівнева клієнт-серверна архітектура (3-tier Client-server)
4. Хмарна клієнт-серверна архітектура (cloud computing)
5. Компонентно-орієнтована архітектура (component-based)
6. Архітектура централізованої бази даних (Database-centric)
7. Подієво-орієнтована архітектра (Event-driven Implicit invocation)
8. Архітектра монолітного додатку (Monolithic application)
9. Peer-to-peer (P2P) архітектура
10. Архітектура обчислювального конвеєру (Pipes and filters)
11. Архітектура плагінів (Plug-ins)
12. REST-архітектура (Representational state transfer)
13. Сервер виконання бізнес-правил (Rule-based)
14. Сервіс-орієнтована архітектура (Service-oriented)
15. Архітектура Shared nothing (нічого не розділяється)
16. Космічно базована архітектура (Space-based)
17. Багаторівнева архітектура (Multitier)

Список 2. Варіанти методологій розробки ПЗ:

1. RAD-методологія.
2. RUP-методологія
3. ICONIX
4. Scrum
5. DYNAMIC SYSTEM DEVELOPMENT METHOD
6. Adaptive Software Development
7. Crystal Clear
8. Feature-Driven Development
9. Pragmatic Programming
10. MSF
11. CANBAN
12. LEAN
13. Agile
14. TDD
15. DDD

Список 3. Зразки предметних областей для проектування та розробки:

1. Інформаційна система для організації та проведення державних закупівель.
2. Система моніторингу міського транспорту
3. ПЗ для ідентифікації апаратного забезпечення та моніторингу встановленого та можливих оновлень драйверів.
4. Система підрахунку та контролю мережевого трафіку при підключенні до мережі через модем.
5. ПЗ для синхронізації паролів між різними апаратними пристроями.

6. Поштовий клієнт (для синхронізації поштових акаунтів).
7. ІС "Кафедра" для автоматизації робіт на кафедрі програмної інженерії.
8. Диспетчерський центр оперативного вирішення скарг мешканців мікрорайону ТОВ ЖКГ «Мрія».
9. Служба обліку спожитої електроенергії в навчальних корпусах університету
10. Автоматизована система обліку звернень та виконаних робіт ОСББ «Ромашка»
11. Автоматизована система обліку газової служби Тернопільської області.
12. АІС Наукова лабораторія університету
13. Система автоматизації центру міжнародної наукової кооперації
14. АІС Студентський сектор університету
15. АІС Архітектурне бюро «Погляд з Парижу»
16. АІС Художня майстерня «Вернісаж»
17. АІС База «Тернопільсир»
18. АІС театру «Багато галасу»
19. Інформаційно довідкова система туристичних пам'яток України
20. Автоматизована інформаційна система реєстрації та обліку замовлень торгівельного підприємства
21. Інформаційна система міста Тернопіль
22. Геоінформаційна система міста Тернопіль
23. Охоронна система ВУЗу
24. Система домашнього енергоменеджменту.
25. Система розподілу та контролю завдаль на підприємстві
26. Інтернет-аукціон.
27. Електронна платіжна система.
28. ПЗ для обміну файлів між різними пристроями.
29. ПЗ для доступу і об'єднання хмарних акаунтів.
30. Ресурс для бронювання та продажу квитків на авто та залізничні перевезення по Україні.
31. Система документообігу підприємства з продаж будівельних матеріалів «Арскераміка».
32. Ресурс для замовлення послуг таксі в місті Тернополі (без привязки до служби таксі, веб-система).
33. Центр прийому замовлень на телерекламу міського телебачення «TV-4»
34. Автоматизація роботи підприємства з ремонту дорожнього покриття та встановлення знаків вуличного руху.
35. Служба реєстрації викликів швидкої допомоги.
36. Служба реєстрації хворих першої міської поліклініки.
37. Готельне господарство «Тернопіль»
38. Центр працевлаштування «Кар'єра»
40. Служба збуту продукції ТМ "Молокія"
41. Магазин «Антикваріат»
42. Система логістики
43. Онлайн-кімната геймера
44. Інформаційна система електронного документообігу на підприємстві
45. Інформаційна система обліку продукції на складі

46. Автоматизована система обліку нарахування заробітної плати
47. Система автоматизації бронювання та продажу квитків на проїзд автобусним транспортом
48. Система автоматизації бронювання та оренди місць на автопарковці.
49. Інформаційна система магазину "Страйкбольного спорядження" (продаж та оренда).
50. Система керування розробкою програмного забезпечення за методологією Scrum.
51. Інформаційна система вело-майстерні
52. Соціальна мережа для студентів/науковців.
53. Система контролю робочого часу працівників.
54. ПЗ для синхронізації акаунтів соціальних мереж
55. ПЗ «Розумний дім».
56. Система реєстрації та обліку замовлень кондитерської фабрики.
57. Служба обліку роботи оздоровчого басейну університету
58. Автоматизована система формування та оцінювання тестових завдань для навчального закладу.
59. Програмний додаток для конвертування зображень у текстовий формат.
60. Система обліку паролів різних ресурсів
61. Система автоматизації роботи страхової агенції
62. Система для цифрового моніторингу здоров'я
63. Автоматизована система ресторану (включаючи сервіс попереднього замовлення).
64. Інформаційна система «Студмістечко»
65. Медична інформаційна система Тернопільської області
66. Система планування та контролю робіт на підприємстві
67. Міська рекламно-інформаційна система
68. Логістична система планування маршруту
69. Робота системи банкоматів банку «Укрсиббанк»

Лабораторна робота 2. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням архітектурного шаблону проектування «Абстрактна фабрика / Abstract Factory»

Тема: Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням архітектурного шаблону проектування «Абстрактна фабрика / Abstract Factory».

Мета роботи: Реалізувати в системі IBM RSA архітектурну модель (АМ) проектування ПЗ - Абстрактна фабрика (AF) з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect.

Теоретичні відомості:

Абстрактна фабрика - гнучка і динамічна архітектурна модель проектування моделі предметної області системи (МПО), що включає взаємозв’язані ієрархії на основі інтерфейсів (абстрактних класів) для створення множини взаємозв’язаних об’єктів, не специфікуючи їх конкретних класів. Спочатку проектуються абстрактні класи (базові класи ієрархій), що описують інтерфейси для створення компонентів системи, а потім створюються похідні конкретні класи, що поліморфно реалізують ці інтерфейси..

Умови застосування :

1. Незалежність системи від способів утворення, компонування і представлення вхідних до неї об'єктів;
2. Класи вхідних колекцій об'єктів взаємозв'язані і мають використовуватися разом;
3. Конфігурування системи ч/з визначену структуру абстрактних класів, що розкривають тільки інтерфейси колекції об'єктів, а не реалізацію.

Переваги:

1. Внесення швидких змін у місця коду ієрархій, строго визначені архітектором (підсистеми/ класи, їх зміна, розширення їх переліку), не порушуючи цілісності всієї системи;
2. Суттєве зменшення кількості класів та їх описів, що веде до оптимізації коду, підвищення рівня читабельності щодо доопрацювання (відхід від важкої спадщини попередників);
3. Забезпечення формалізованих уніфікованих запитів користувача до системи ч/з інтерфейси (відокремлення описів класів від реалізацій, інкапсуляція і захист від несанкціонованого доступу);
4. Гнучкість і незалежність розроблюваних компонент ПЗ, їх взаємозамінність, повторне використання коду.

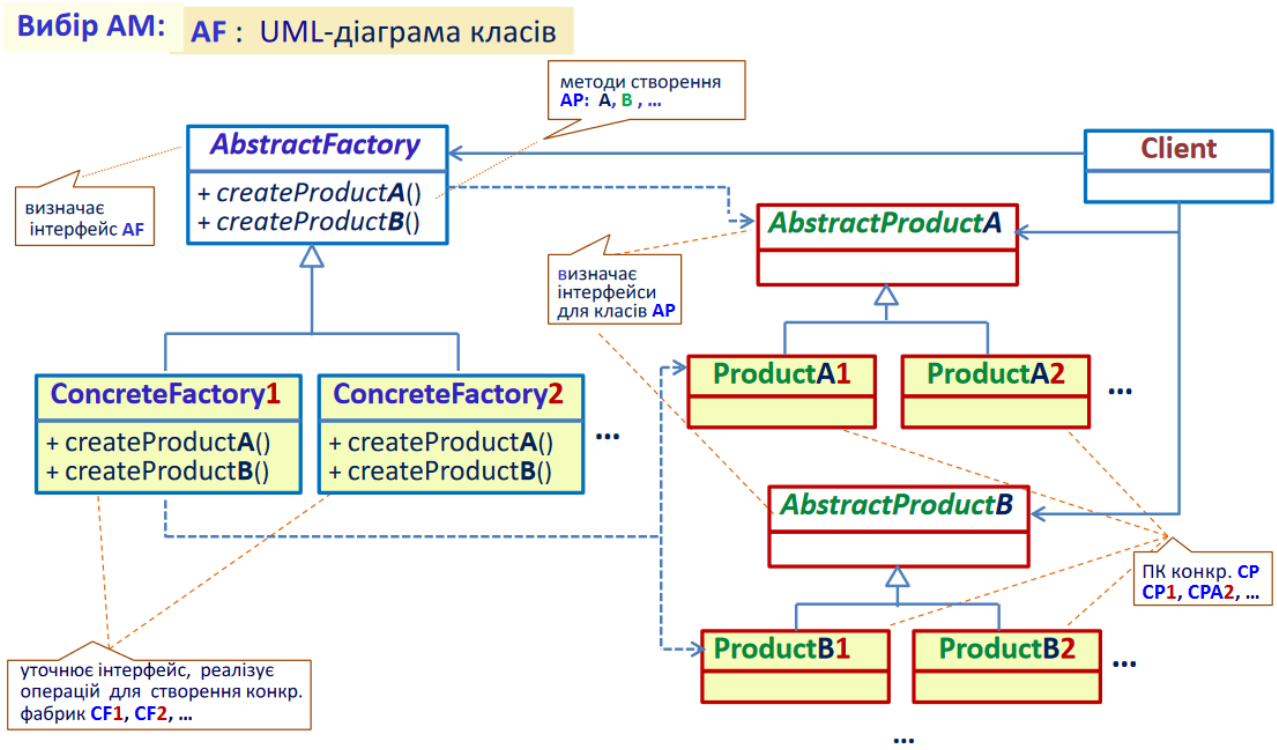

Рис. 2.1. Діаграма класів АМ «Абстрактна фабрика»

Завдання роботи:

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування Абстрактна фабрика (Abstract Factory).

2. Визначити вимоги до ПЗ згідно обраної предметної області, проаналізувати прецеденти, способи застосування та архітектурну модель майбутньої системи. Побудувати діаграму ВВ. Продумати архітектуру класів майбутніх сутностей, відношенням між ними.

3. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на шаблонах та Абстрактна фабрика (Abstract Factory). Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.

4. Опишіть спосіб застосування середовища проектування RSA з покроковою інструкцією.

5. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування. Добавити коментарі та нотації.

6. Підготувати звіт до захисту лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Звіт до лабораторної роботи повинен містити:

- Загальний опис архітектурної моделі ПЗ «Абстрактна фабрика», Умови її застосування та переваги;

- Аналіз предметної області. Опис способу та умов застосування розглянутого шаблону проектування.

- Модель варіантів використання (в IBM RSA) обраного прецеденту (модуля програми). Use Case діаграма.

- Діаграму класів (в рамках даної моделі, лише необхідний мінімум класів).

- Описи класів та їх атрибутів, методів. Опис зв'язків та залежностей між класами.

- Розширений опис основних етапів процесу створення проектування та способу застосування інструментального середовища проектування IBM RSA. Описи основних меню IBM RSA та послідовності їх застосування при створенні проекту і побудові основних діаграм.

- Програмну реалізація: Компонентна модель (опис структури коду);

- код автоматично згенерований по діаграмі класів в IBM RSA, мова C++ або Java;

- код розбитий на модулі/ компоненти, містить коментарі;

- в оформленні коду дотримуватись єдиного стилю та доповнювати код коментарями.

- Основні методи і сценарії способу застосування даної реалізації показати в діаграмі послідовності (sequence diagram)

- Описати можливість, переваги та недоліки застосування даного шаблону Абстрактна фабрика для даної системи предметної області

- Висновки щодо виконання лабораторної роботи.

Лабораторна робота 3. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Декоратор (Picasso)» та «Пристосуванець (Flyweight)»

Тема*:* Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Декоратор (Picasso)» та «Пристосуванець (Flyweight)».

Мета роботи:
Реалізувати в системі IBM RSA архітектурні моделі (АМ) ПЗ - «Декоратор (Picasso)» та «Пристосуванець (Flyweight)» з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect для обраної предметної області.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Декоратор (Picasso)».
2. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Декоратор (Picasso)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
3. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Декоратор. Добавити коментарі та нотації.
4. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Пристосуванець (Flyweight)».
5. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Пристосуванець (Flyweight)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
6. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Пристосуванець. Добавити коментарі та нотації.
7. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Звіт до лабораторної роботи повинен містити:

- Аналіз предметної області. Опис способу та умов застосування розглянутого шаблону проектування.
- Модель варіантів використання (в IBM RSA) обраного прецеденту (модуля програми). Use Case діаграма.
- Діаграму класів (в рамках даної моделі, лише необхідний мінімум класів).
- Описи класів та їх атрибутів, методів. Опис зв’язків та залежностей між класами.
- Програмну реалізація: Компонентна модель (опис структури коду)

- код автоматично згенерований по діаграмі класів в IBM RSA, мова C++ або Java;
- код розбитий на модулі/ компоненти;
- в оформленні коду дотримуватись єдиного стилю [5] та доповнювати код коментарями.

- Основні методи і сценарії способу застосування даної реалізації показати в діаграмі послідовності (sequence diagram)
- Описати можливість, переваги та недоліки застосування даного шаблону для даної системи предметної області
- Висновки щодо виконання лабораторної роботи.

Теоретичні відомості

Структурні патерни проектування відповідають за побудову зручних в підтримці ієрархій класів та визначають взаємозалежності між класами і об'єктами, дозволяючи їм працювати спільно. При цьому можуть використовуватися такі механізми компонування системи на основі класів і об'єктів:

- Наслідування, коли клас визначає інтерфейс, а підкласи - реалізацію. Структури на основі наслідування виходять статичними.
- Композиція, коли структури будуються шляхом об'єднання об'єктів деяких класів. Композиція дозволяє отримувати структури, які можна змінювати під час виконання.

3.1. Декоратор — це структурний патерн проектування, що дає змогу динамічно додавати об’єктам нову функціональність, загортаючи їх у корисні «обгортки». Спадкування — це перше, що приходить в голову багатьом програмістам, коли потрібно розширити яку-небудь чинну поведінку. Проте механізм спадкування має кілька прикрих проблем.

- Він статичний. Ви не можете змінити поведінку об’єкта, який вже існує. Для цього необхідно створити новий об’єкт, вибравши інший підклас
- Він не дозволяє наслідувати поведінку декількох класів одночасно. Тому доведеться створювати безліч підкласів-комбінацій, щоб досягти поєднання поведінки.

Одним зі способів, що дозволяє обійти ці проблеми — є заміна спадкування агрегацією або композицією . Це той випадок, коли один об’єкт утримує інший і делегує йому роботу, замість того, щоб самому успадкувати його поведінку. Саме на цьому принципі побудовано патерн Декоратор. Декоратор має альтернативну назву — обгортка. Вона більш вдало описує суть патерна: ви розміщуєте цільовий об’єкт у іншому об’єкті-обгортці, який запускає базову поведінку об’єкта, а потім додає до результату щось своє.

Обидва об’єкти мають загальний інтерфейс, тому для користувача немає жодної різниці, з чим працювати — з чистим чи загорнутим об’єктом. Ви можете використовувати кілька різних обгорток одночасно — результат буде мати об’єднану поведінку всіх обгорток. Застосовуючи Декоратор, ви не змінюєте початковий клас і не створюєте дочірніх класів.

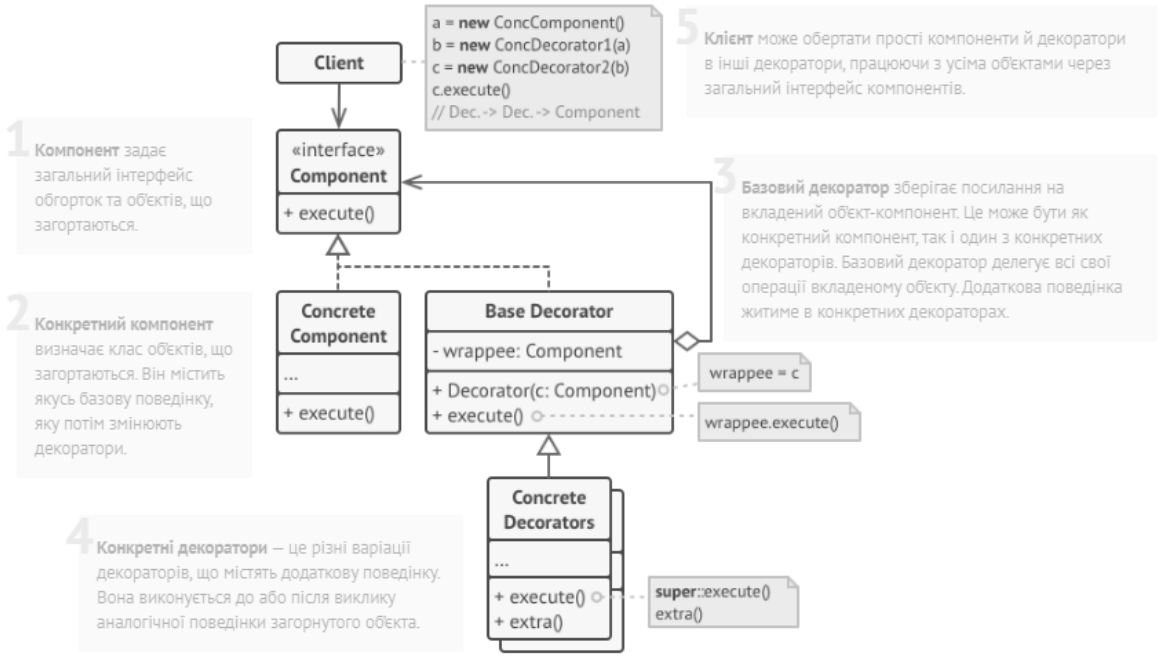

Рис. 3.1. Структура архітектури класів для шаблону Декоратор [6]

Застосування:

- Якщо вам потрібно додавати об’єктам нові обов’язки «на льоту», непомітно для коду, який їх використовує. Об’єкти вкладаються в обгортки, які мають додаткові поведінки. Обгортки і самі об’єкти мають однаковий інтерфейс, тому клієнтам не важливо, з чим працювати — зі звичайним об’єктом чи з загорнутим.
- Якщо не можна розширити обов’язки об’єкта за допомогою спадкування. У багатьох мовах програмування є ключове слово final, яке може заблокувати спадкування класу. Розширити такі класи можна тільки за допомогою Декоратора.

Переваги та Недоліки:

Переваги	Недоліки
• Більша гнучкість, ніж у спадкування. • Дозволяє додавати обов’язки «на льоту». • Можна додавати кілька нових обов’язків одразу. • Дозволяє мати кілька дрібних об’єктів, замість одного об’єкта «на всі випадки життя».	• Важко конфігурувати об’єкти, які загорнуто в декілька обгорток одночасно. • Велика кількість крихітних класів.

3.2. Легковаговик — це структурний патерн проектування, що дає змогу вмістити більшу кількість об’єктів у відведеній оперативній пам’яті. Легковаговик заощаджує пам’ять, розподіляючи спільний стан об’єктів між собою, замість зберігання однакових даних у кожному об’єкті.

Патерн Легковаговик пропонує не зберігати зовнішній стан у класі, а передавати його до тих чи інших методів через параметри. Таким чином, одні і ті самі об’єкти можна буде повторно використовувати в різних контекстах. Головна ж перевага в тому, що тепер знадобиться набагато менше об’єктів, адже вони тепер відрізнятимуться тільки внутрішнім станом, а він не має так багато варіацій.

Патерн Легковаговик слід застосовувати при дотриманні всіх наступних умов:

- Коли додаток використовує велику кількість одноманітних об'єктів, через що відбувається виділення великої кількості пам'яті
- Коли частина стану об'єкта, яке є змінним, можна винести за. Винесення зовнішнього стану дозволяє замінити безліч об'єктів невеликою групою загальних поділюваних об'єктів.

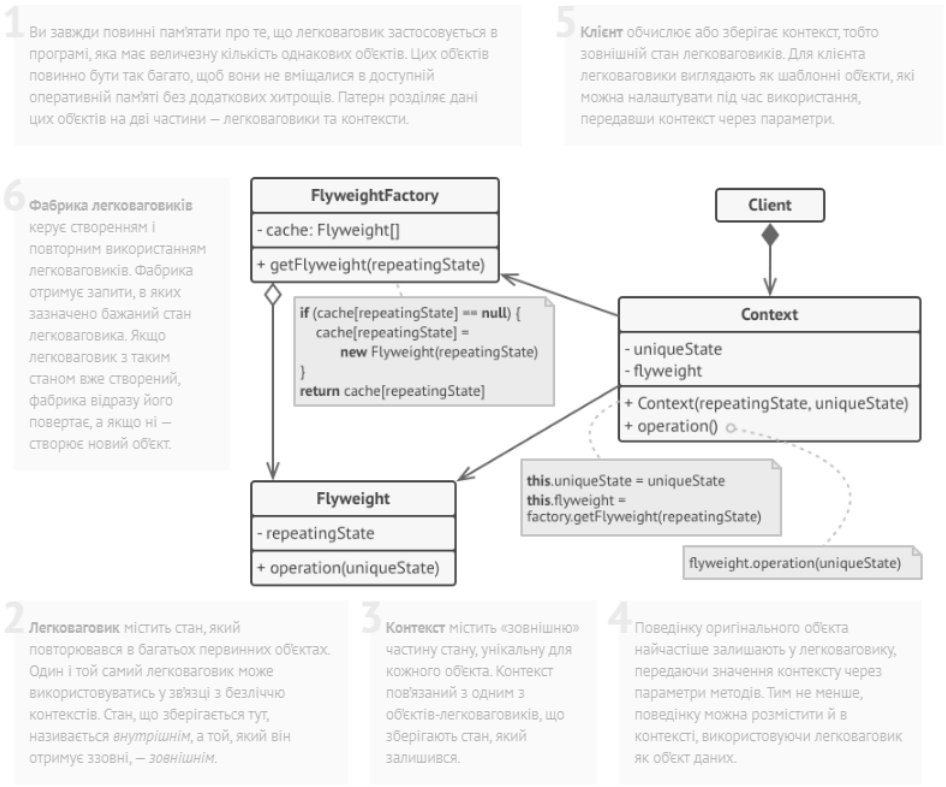

Рис. 3.1. Структура архітектури класів для шаблону Легковаговик [7]

В якості стандартного застосування даного патерну можна навести такий приклад. Текст складається з окремих символів. Кожен символ може зустрічатися на одній сторінці тексту багато разів. Однак в комп'ютерній програмі було б занадто накладно виділяти пам'ять для кожного окремого символу в тексті. Набагато простіше було б визначити повний набір символів, наприклад, у вигляді таблиці з 128 знаків (алфавітно-цифрові символи в

різних регістрах, розділові знаки і т.д.). А в тексті застосувати цей набір загальних поділюваних символів, замість сотень і тисяч об'єктів, які могли б використовуватися в тексті. І як наслідок такого підходу буде зменшення кількості використовуваних об'єктів і зменшення використовуваної пам'яті.

Застосування:

- Якщо не вистачає оперативної пам’яті для підтримки всіх потрібних об’єктів.
  Ефективність патерна Легковаговик багато в чому залежить від того, як і де він використовується. Застосовуйте цей патерн у випадках, коли виконано всі перераховані умови:
  - у програмі використовується велика кількість об’єктів;
  - через це високі витрати оперативної пам’яті;
  - більшу частину стану об’єктів можна винести за межі їхніх класів;
  - великі групи об’єктів можна замінити невеликою кількістю об’єктів, що розділяються, оскільки зовнішній стан винесено.

Переваги та Недоліки:

Переваги	Недоліки
• Заощаджує оперативну пам’ять	• Витрачає процесорний час на пошук/обчислення контексту. • Ускладнює код програми внаслідок введення безлічі додаткових класів

Лабораторна робота 4. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Замісник (Proxy)» та «Фасад (Facade)»

Тема: Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Замісник (Proxy)» та «Фасад (Facade)».

Мета роботи:

Реалізувати в системі IBM RSA архітектурні моделі (АМ) ПЗ - «Замісник (Proxy)» та «Фасад (Facade)» з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect для обраної предметної області.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Замісник (Proxy)».
2. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Замісник (Proxy)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
3. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Замісник. Добавити коментарі та нотації.
4. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Фасад (Facade)».
5. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Фасад (Facade)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
6. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Фасад. Добавити коментарі та нотації.
7. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості

4.1. Замісник — це структурний патерн проектування, що дає змогу підставляти замість реальних об’єктів спеціальні об’єкти-замінники. Ці об’єкти перехоплюють виклики до оригінального об’єкта, дозволяючи зробити щось до чи після передачі виклику оригіналові. Це - об’єкт, який виступає прошарком між клієнтом та реальним сервісним об’єктом. Замісник отримує виклики від клієнта, виконує свою функцію (контроль доступу, кешування, зміна запиту та інше), а потім передає виклик сервісному об’єктові.

Патерн Замісник пропонує створити новий клас-дублер, який має той самий інтерфейс, що й оригінальний службовий об’єкт. При отриманні запиту від клієнта об’єкт-замісник сам би створював примірник службового об’єкта та переадресовував би йому всю реальну роботу.

Замісник має той самий інтерфейс, що і реальний об’єкт, тому для клієнта немає різниці — працювати з реальним об’єктом безпосередньо, чи за допомогою замісника.

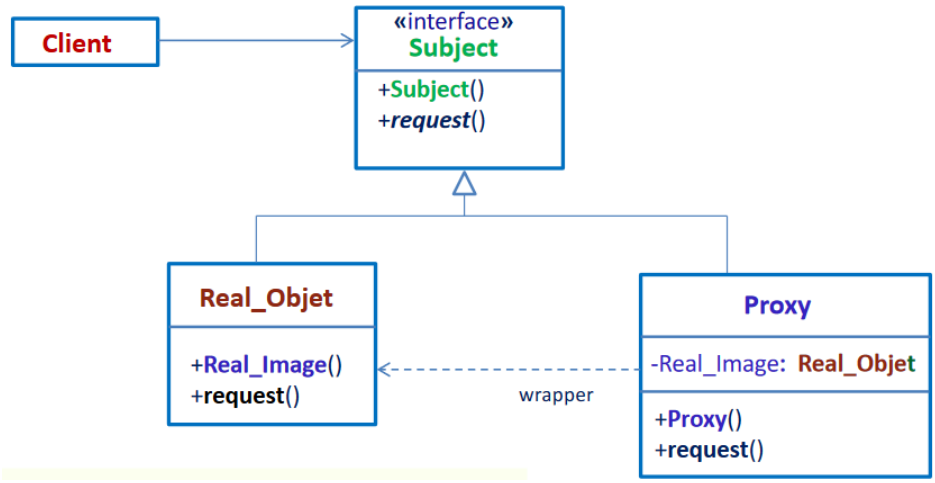

Рис. 4.1. Діаграма класів шаблону Замісник

Застосування:

- Лінива ініціалізація (віртуальний проксі). Коли у вас є важкий об’єкт, який завантажує дані з файлової системи або бази даних.

  Замість того, щоб завантажувати дані відразу після старту програми, можна заощадити ресурси й створити об’єкт тоді, коли він дійсно знадобиться.
- Захист доступу (захищаючий проксі). Коли в програмі є різні типи користувачів, і вам хочеться захистити об’єкт від неавторизованого доступу. Наприклад, якщо ваші об’єкти — це важлива частина операційної системи, а користувачі — сторонні програми (корисні чи шкідливі).

  Проксі може перевіряти доступ під час кожного виклику та передавати виконання службовому об’єкту, якщо доступ дозволено.
- Локальний запуск сервісу (віддалений проксі). Коли справжній сервісний об’єкт знаходиться на віддаленому сервері.

  У цьому випадку замісник транслює запити клієнта у виклики через мережу по протоколу, який є зрозумілим віддаленому сервісу.
- Логування запитів (логуючий проксі). Коли потрібно зберігати історію звернень до сервісного об’єкта.

  Замісник може зберігати історію звернення клієнта до сервісного об’єкта.
- Кешування об’єктів («розумне» посилання). Коли потрібно кешувати результати запитів клієнтів і керувати їхнім життєвим циклом.

  Замісник може підраховувати кількість посилань на сервісний об’єкт, які були віддані клієнту та залишаються активними. Коли всі посилання звільняться, можна буде звільнити і сам сервісний об’єкт (наприклад, закрити підключення до бази даних).

  Крім того, Замісник може відстежувати, чи клієнт не змінював сервісний об’єкт. Це дозволить повторно використовувати об’єкти й суттєво заощаджувати ресурси, особливо якщо мова йде про великі «ненажерливі» сервіси.

Переваги та Недоліки:

Переваги	Недоліки
• Дозволяє контролювати сервісний об’єкт непомітно для клієнта. • Може працювати, навіть якщо сервісний об’єкт ще не створено. • Може контролювати життєвий цикл службового об’єкта.	• Ускладнює код програми внаслідок введення додаткових класів. • Збільшує час отримання відклику від сервісу.

4.2. Фасад — це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. Фасад може бути спрощеним відображенням системи, що не має 100%% тієї функціональності, якої можна було б досягти, використовуючи складну підсистему безпосередньо. Разом з тим, він надає саме ті «фічі», які потрібні клієнтові, і приховує все інше.

Фасад корисний у тому випадку, якщо ви використовуєте якусь складну бібліотеку з безліччю рухомих частин, з яких вам потрібна тільки частина.

Вашому коду доводиться працювати з великою кількістю об’єктів певної складної бібліотеки чи фреймворка. Ви повинні самостійно ініціалізувати ці об’єкти, стежити за правильним порядком залежностей тощо. В результаті бізнес-логіка ваших класів тісно переплітається з деталями реалізації сторонніх класів. Такий код досить складно розуміти та підтримувати.

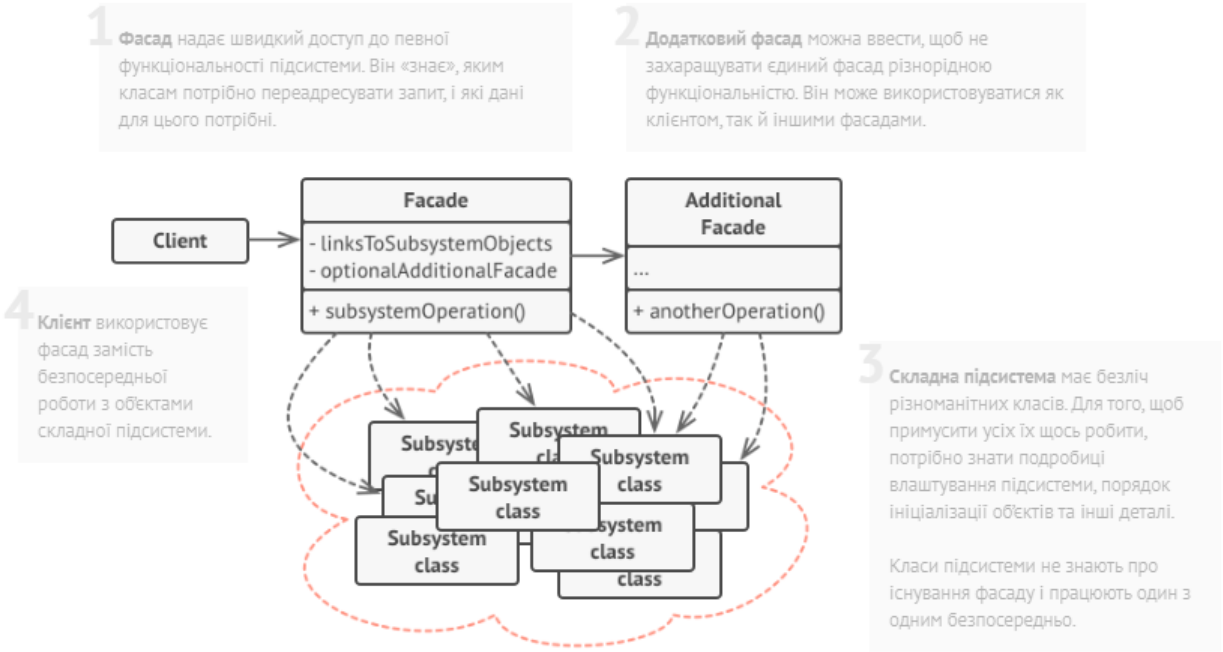

Рис. 4.2. Структура архітектури класів для шаблону Фасад [8]

Застосування:

- Якщо вам потрібно надати простий або урізаний інтерфейс до складної підсистеми.

  Часто підсистеми ускладнюються в міру розвитку програми. Застосування більшості патернів призводить до появи менших класів, але у великій кількості. Таку підсистему простіше використовувати повторно, налаштовуючи

її кожен раз під конкретні потреби, але, разом з тим, використовувати таку підсистему без налаштовування важче. Фасад пропонує певний вид системи за замовчуванням, який влаштовує більшість клієнтів.

- Якщо ви хочете розкласти підсистему на окремі рівні.

  Використовуйте фасади для визначення точок входу на кожен рівень підсистеми. Якщо підсистеми залежать одна від одної, тоді залежність можна спростити, дозволивши підсистемам обмінюватися інформацією тільки через фасади. Наприклад, візьмемо ту ж саму складну систему конвертації відео. Ви хочете розбити її на окремі шари для роботи з аудіо й відео. Можна спробувати створити фасад для кожної з цих частин і примусити класи аудіо та відео обробки спілкуватися один з одним через ці фасади, а не безпосередньо.

Переваги та Недоліки:

Переваги	Недоліки
• Ізолює клієнтів від компонентів складної підсистеми.	• Фасад ризикує стати божественим об’єктом, прив’язаним до всіх класів програми.

Лабораторна робота 5. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Спостерігач (Observer)» та «Відвідувач (Visitor)»

Тема: Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Спостерігач (Observer)» та «Відвідувач (Visitor)».

Мета роботи:

Реалізувати в системі IBM RSA архітектурні моделі (АМ) ПЗ - «Спостерігач (Observer)» та «Відвідувач (Visitor)» з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect для обраної предметної області.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Спостерігач (Observer)».
2. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Спостерігач (Observer)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
3. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Спостерігач. Добавити коментарі та нотації.
4. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Відвідувач (Visitor)».
5. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Відвідувач (Visitor)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
6. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Відвідувач. Добавити коментарі та нотації.
7. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості

5.1. Спостерігач — це поведінковий патерн проектування, який створює механізм підписки, що дає змогу одним об’єктам стежити й реагувати на події, які відбуваються в інших об’єктах**.** Список підписників складається динамічно, об’єкти можуть як підписуватися на певні події, так і відписуватися від них прямо під час виконання програми.

Для додавання до програми нових підписників не потрібно змінювати класи видавців, допоки вони працюють із підписниками через загальний інтерфейс.

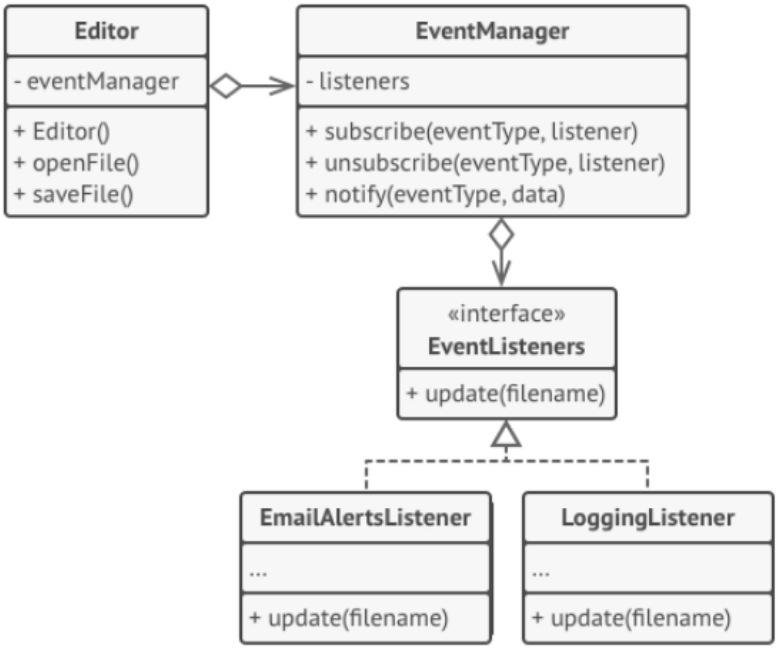

Рис. 5.1. Структура архітектури класів для шаблону Спостерігач [9]

Застосування:

Якщо після зміни стану одного об’єкта потрібно щось зробити в інших, але ви не знаєте наперед, які саме об’єкти мають відреагувати. Патерн Спостерігач надає змогу будь-якому об’єкту з інтерфейсом підписника зареєструватися для отримання сповіщень про події, що трапляються в об’єктах-видавцях.

Якщо одні об’єкти мають спостерігати за іншими, але тільки у визначених випадках. Видавці ведуть динамічні списки. Усі спостерігачі можуть підписуватися або відписуватися від отримання сповіщень безпосередньо під час виконання програми

Переваги та Недоліки:

Переваги	Недоліки
Видавці не залежать від конкретних класів підписників і навпаки. Ви можете підписувати і відписувати одержувачів «на льоту». Реалізує принцип відкритості/закритості.	Підписники сповіщуються у випадковій послідовності.

5.2. Відвідувач — це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об’єктів, над якими ці операції можуть виконуватися**.** Патерн Відвідувач пропонує розмістити нову поведінку в окремому класі, замість того, щоб множити її відразу в декількох класах. Об’єкти, з якими повинна бути пов’язана поведінка, не виконуватимуть її самостійно. Замість цього ви будете передавати ці об’єкти до методів відвідувача.

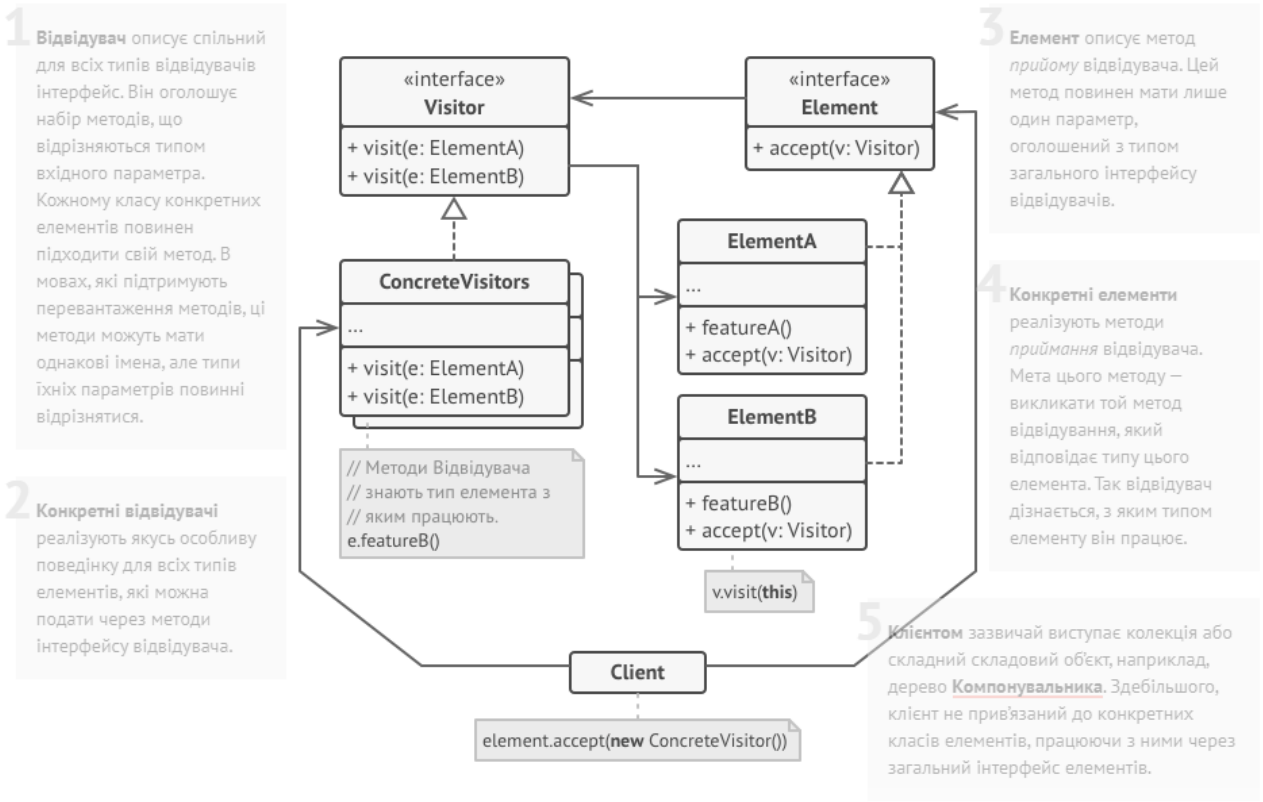

Рис. 5.2. Структура архітектури класів для шаблону Відвідувач [10]

Застосування:

Якщо вам потрібно виконати якусь операцію над усіма елементами складної структури об’єктів, наприклад, деревом. Відвідувач дозволяє застосовувати одну і ту саму операцію до об’єктів різних класів.

Якщо над об’єктами складної структури об’єктів потрібно виконувати деякі не пов’язані між собою операції, але ви не хочете «засмічувати» класи такими операціями. Відвідувач дозволяє витягти споріднені операції з класів, що складають структуру об’єктів, помістивши їх до одного класу-відвідувача. Якщо структура об’єктів використовується в декількох програмах, то патерн дозволить кожній програмі мати тільки потрібні в ній операції.

Якщо нова поведінка має сенс тільки для деяких класів з існуючої ієрархії. Відвідувач дозволяє визначити поведінку тільки для цих класів, залишивши її порожньою для всіх інших.

Переваги та Недоліки:

Переваги	Недоліки
Спрощує додавання операцій, працюючих зі складними структурами об’єктів. Об’єднує споріднені операції в одному класі. Відвідувач може накопичувати стан при обході структури елементів..	Патерн невиправданий, якщо ієрархія елементів часто змінюється. Може призвести до порушення інкапсуляції елементів.

Лабораторна робота 6. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Стратегія(Strategy)» та «Ланцюг відповідальності (Chain of Responsibility)»

Тема: Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням структурних шаблонів проектування типу «Стратегія(Strategy)» та «Ланцюг відповідальності (Chain of Responsibility)».

Мета роботи*:*

Реалізувати в системі IBM RSA архітектурні моделі (АМ) ПЗ - «Стратегія(Strategy)» та «Ланцюг відповідальності (Chain of Responsibility)» з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect для обраної предметної області.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Стратегія(Strategy)».
2. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Стратегія(Strategy)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
3. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Стратегія. Добавити коментарі та нотації.
4. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Ланцюг відповідальності (Chain of Responsibility)».
5. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Ланцюг відповідальності (Chain of Responsibility)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
6. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Ланцюг відповідальності. Добавити коментарі та нотації.
7. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості

6.1. Стратегія — це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми. Патерн Стратегія пропонує визначити сімейство схожих алгоритмів, які часто змінюються або розширюються, й винести їх до власних класів, які називають стратегіями.

Важливо, щоб всі стратегії мали єдиний інтерфейс. Використовуючи цей інтерфейс, контекст буде незалежним від конкретних класів стратегій. З іншого боку, ви зможете змінювати та додавати нові види алгоритмів, не чіпаючи код контексту.

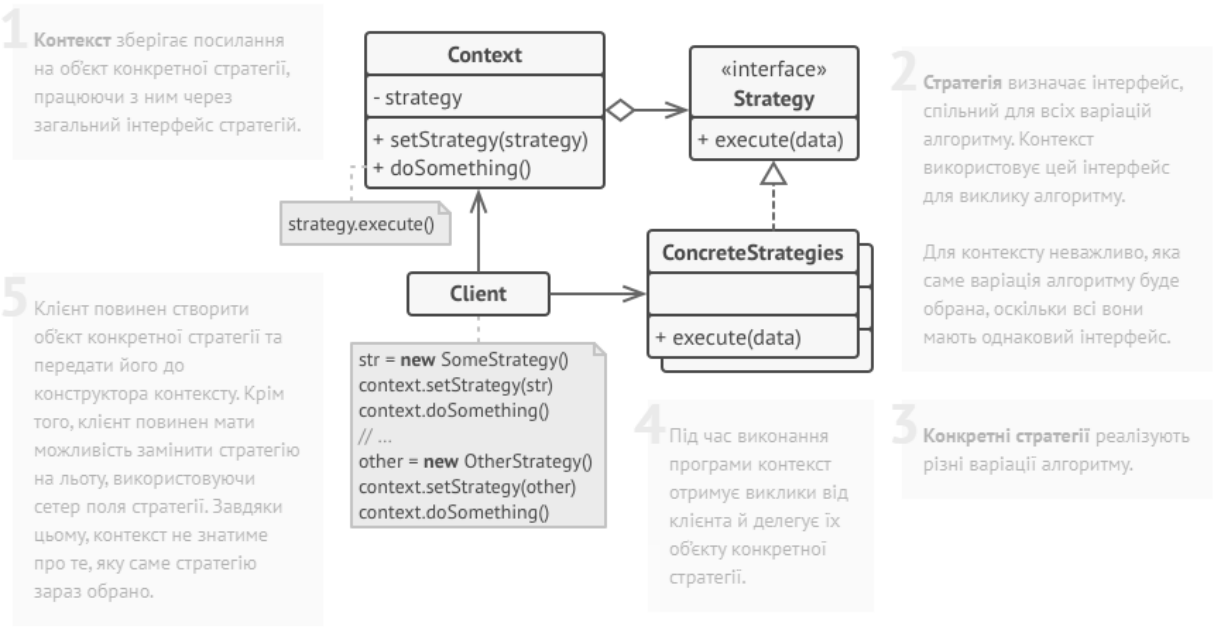

Рис. 6.1. Структура архітектури класів для шаблону Стратегія [11]

Застосування:

- Якщо вам потрібно використовувати різні варіації якого-небудь алгоритму всередині одного об’єкта. Стратегія дозволяє варіювати поведінку об’єкта під час виконання програми, підставляючи до нього різні об’єкти-поведінки (наприклад, що відрізняються балансом швидкості та споживання ресурсів)
- Якщо у вас є безліч схожих класів, які відрізняються лише деякою поведінкою. Стратегія дозволяє відокремити поведінку, що відрізняється, у власну ієрархію класів, а потім звести початкові класи до одного, налаштовуючи його поведінку стратегіями.
- Якщо ви не хочете оголювати деталі реалізації алгоритмів для інших класів. Стратегія дозволяє ізолювати код, дані й залежності алгоритмів від інших об’єктів, приховавши ці деталі всередині класів-стратегій
- Якщо різні варіації алгоритмів реалізовано у вигляді розлогого умовного оператора. Кожна гілка такого оператора є варіацією алгоритму. Стратегія розміщує кожну лапу такого оператора до окремого класу-стратегії. Потім контекст отримує певний об’єкт-стратегію від клієнта й делегує йому роботу. Якщо раптом знадобиться змінити алгоритм, до контексту можна подати іншу стратегію

Переваги та Недоліки:

Переваги	Недоліки
• Гаряча заміна алгоритмів на льоту. • Ізолює код і дані алгоритмів від інших класів. • Заміна спадкування делегуванням. • Реалізує принцип відкритості/закритості.	• Ускладнює програму внаслідок додаткових класів. • Клієнт повинен знати, в чому полягає різниця між стратегіями, щоб вибрати потрібну.

6.2. Ланцюг відповідальності (Chain of Responsibility) — це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен

наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком. Як і багато інших поведінкових патернів, ланцюжок обов'язків базується на тому, щоб перетворити окремі поведінки на об'єкти. У нашому випадку кожна перевірка переїде до окремого класу з одним методом виконання. Дані запиту, що перевіряється, передаватимуться до методу як аргументи.

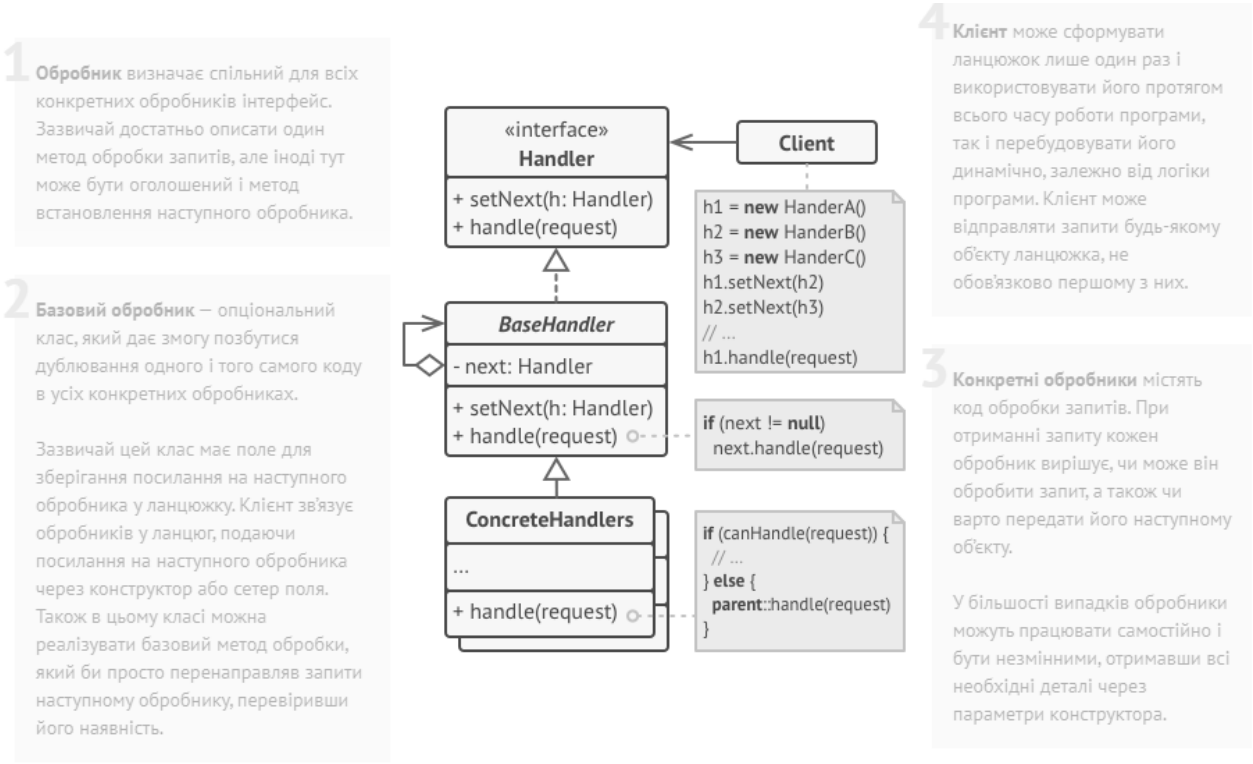

Рис. 6.2. Структура архітектури класів для шаблону Ланцюг відповідальності [12]

Застосування:

- Якщо програма має обробляти різноманітні запити багатьма способами, але заздалегідь невідомо, які конкретно запити надходитимуть і які обробники для них знадобляться. За допомогою Ланцюжка обов'язків ви можете зв'язати потенційних обробників в один ланцюг і по отриманню запита по черзі питати кожного з них, чи не хоче він обробити даний запит.
- Якщо важливо, щоб обробники виконувалися один за іншим у суворому порядку. Ланцюжок обов'язків дозволяє запускати обробників один за одним у тій послідовності, в якій вони стоять в ланцюзі.
- Якщо набір об'єктів, здатних обробити запит, повинен задаватися динамічно. У будь-який момент ви можете втрутитися в існуючий ланцюжок і перевизначити зв'язки так, щоби прибрати або додати нову ланку.

Переваги та Недоліки:

Переваги	Недоліки
• Зменшує залежність між клієнтом та обробниками. • Реалізує принцип єдиного обов'язку. • Реалізує принцип відкритості/закритості	• Запит може залишитися ніким не опрацьованим.

Лабораторна робота 7. Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням поведінкового шаблону проектування типу «Стан(State)» та структурного шаблону «Компонувальник(Composite)»

Тема: Побудова архітектури ПЗ обраної предметної області в середовищі IBM RSA з використанням поведінкового шаблону проектування типу «Стан(State)» та структурного шаблону «Компонувальник(Composite)».

Мета роботи*:*

Реалізувати в системі IBM RSA архітектурні моделі (АМ) ПЗ - «Стан(State)» та «Компонувальник(Composite)» з використанням інструментальних засобів розробки ПЗ IBM Rational Software Architect для обраної предметної області.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Стан(State)».
2. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Стан(State)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
3. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Стан. Добавити коментарі та нотації.
4. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурного шаблону проектування «Компонувальник(Composite)».
5. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б базувалася на патерні «Компонувальник(Composite)». Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів.
6. Згенерувати програмний код реалізації спроектованої системи згідно розглянутого шаблону проектування Компонувальник. Добавити коментарі та нотації.
7. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості

7.1. Стан — це поведінковий патерн проектування, що дає змогу об’єктам змінювати поведінку в залежності від їхнього стану. Ззовні створюється враження, ніби змінився клас об’єкта. Патерн Стан неможливо розглядати у відриві від концепції машини станів, також відомої як стейт-машина або кінцевий автомат. Основна ідея в тому, що програма може знаходитися в одному з кількох станів, які увесь час змінюють один одного. Набір цих станів, а також переходів між ними, визначений наперед та кінцевий. Перебуваючи в різних станах, програма може по-різному реагувати на одні і ті самі події, що відбуваються з нею.

Машину станів найчастіше реалізують за допомогою множини умовних операторів, if або switch, які перевіряють поточний стан об’єкта та виконують відповідну поведінку. Патерн Стан пропонує створити окремі класи для кожного стану, в якому може перебувати контекстний об’єкт, а потім винести туди поведінки, що відповідають цим станам.

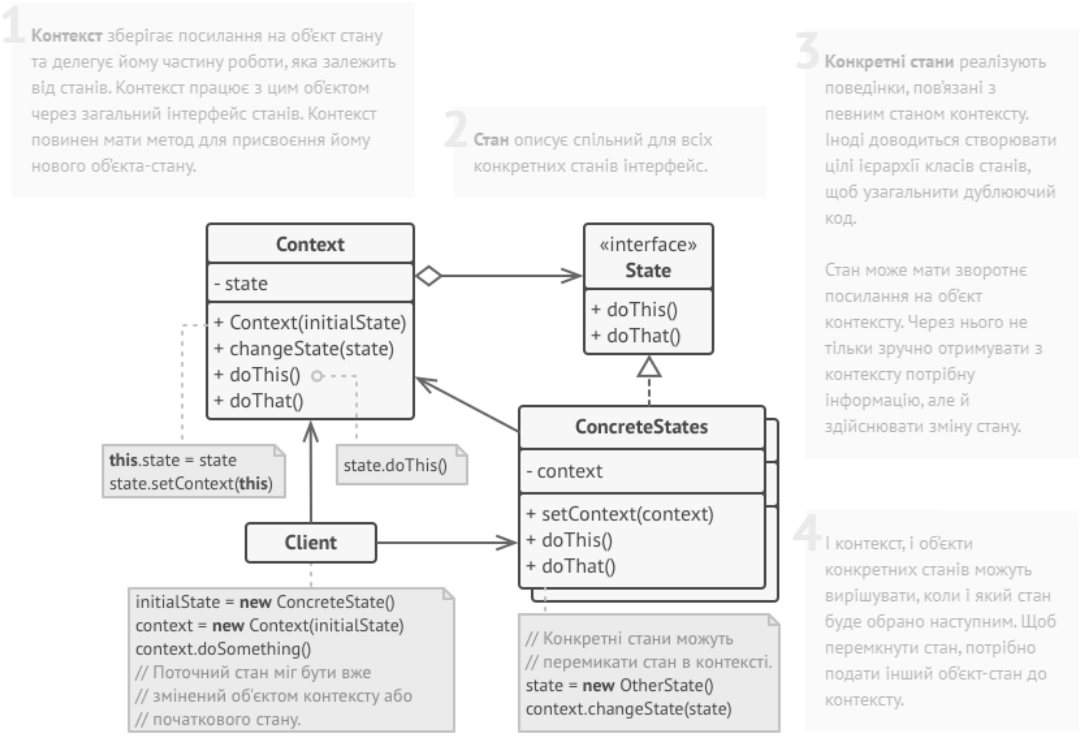

Рис. 7.1. Структура архітектури класів для шаблону Стан [13]

Застосування:

- Якщо у вас є об’єкт, поведінка якого кардинально змінюється в залежності від внутрішнього стану, причому типів станів багато, а їхній код часто змінюється. Патерн пропонує виділити в окремі класи всі поля й методи, пов’язані з визначеним станом. Початковий об’єкт буде постійно посилатися на один з об’єктів-станів, делегуючи йому частину своєї роботи. Для зміни стану до контексту достатньо буде підставляти інший об’єкт-стан.
- Якщо код класу містить безліч великих, схожих один на одного умовних операторів, які вибирають поведінки в залежності від поточних значень полів класу. Патерн пропонує перемістити кожну гілку такого умовного оператора до власного класу. Сюди ж можна поселити й усі поля, пов’язані з цим станом.
- Якщо ви свідомо використовуєте табличну машину станів, побудовану на умовних операторах, але змушені миритися з дублюванням коду для схожих станів та переходів. Патерн Стан дозволяє реалізувати ієрархічну машину станів, що базується на наслідуванні. Ви можете успадкувати схожі стани від одного батьківського класу та винести туди весь дублюючий код.

Переваги та Недоліки:

Переваги	Недоліки
• Позбавляє від безлічі великих умовних операторів машини станів. • Концентрує в одному місці код, пов’язаний з певним станом. • Спрощує код контексту.	• Може невиправдано ускладнити код, якщо станів мало, і вони рідко змінюються.

7.2. Компонувальник (Composite) — це структурний патерн проектування, що дає змогу згрупувати декілька об’єктів у деревоподібну структуру, а потім працювати з нею так, ніби це одиничний об’єкт. Патерн Компонувальник має сенс тільки в тих випадках, коли основна модель вашої програми може бути структурована у вигляді дерева.

Структура архітектури класів для шаблону **Компонувальник**:

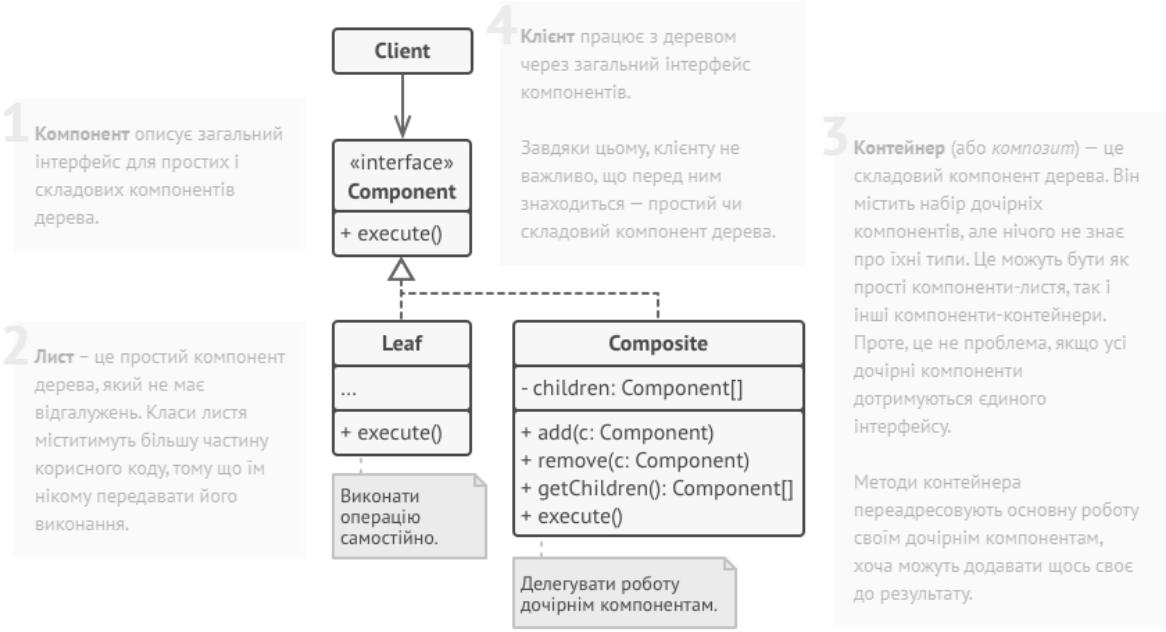

Рис. 7.2. Структура архітектури класів для шаблону Компонувальник [14]

Застосування:

- Якщо вам потрібно представити деревоподібну структуру об’єктів. Патерн Компонувальник пропонує зберігати в складових об’єктах посилання на інші прості або складові об’єкти. Вони, у свою чергу, теж можуть зберігати свої вкладені об’єкти і так далі. У підсумку, ви можете будувати складну деревоподібну структуру даних, використовуючи всього два основних різновида об’єктів
- Якщо клієнти повинні однаково трактувати прості та складові об’єкти. Завдяки тому, що прості та складові об’єкти реалізують спільний інтерфейс, клієнту байдуже, з яким саме об’єктом він працюватиме.

Переваги та Недоліки:

Переваги	Недоліки
• Спрощує архітектуру клієнта при роботі зі складним деревом компонентів. • Полегшує додавання нових видів компонентів.	• Створює занадто загальний дизайн класів.

Лабораторна робота 8. Проектування та реалізація програмного продукту з використанням архітектурної моделі «Клієнт-Сервер»

Тема: Проектування та реалізація програмного продукту з використанням архітектурної моделі «Клієнт-Сервер».

Мета роботи:

Ознайомитися з реалізацією та специфікою застосування архітектури «клієнт-сервер». Спроектувати програмну систему (або її невелику частину), архітектура якої б відповідала моделі ПЗ «Клієнт-Сервер». Проектувати клієнтську частину програмного продукту (клієнтський додаток або застосунок) та API серверної частини для взаємодії. Реалізувати спроектовану систему з використанням однієї з мов програмування на вибір відповідно до спроектованої архітектури.

Завдання роботи

1. Опрацюйте теоретичний матеріал. Ознайомтесь з специфікою архітектурної моделі «Клієнт-Сервер».
2. Розробити та описати діаграму **варіантів використання** для клієнтської частини програмної системи. Описати базовий функціонал системи.
3. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б відповідала **клієнтській** частині розглянутої архітектури для обраної предметної області. Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів, зв’язки між класами.
4. Спроектувати **діаграму послідовності** для одного з фловів використання клієнтської частини програмної системи.
5. Згенерувати програмний код реалізації спроектованої системи згідно спроектованої архітектури **клієнтської частини**. Стан. Добавити коментарі та нотації.
6. Ознайомитися з Терміном та базовою реалізацією API серверної частини архітектури «Клієнт-Сервер» для взаємодії з клієнтською частиною.
7. Розробити та описати діаграму **варіантів використання** для API-серверної частини програмної системи. Описати базовий функціонал API.
8. Спроектувати у середовищі IBM Rational Software Architect власну ієрархію класів, яка б відповідала частині **API** розглянутої архітектури для обраної предметної області. Описати та прокоментувати використання ієрархії класів, інтерфейси, розмежування атрибутів та методів конкретних класів, зв’язки між класами.
9. Спроектувати **діаграму послідовності** для одного API, що використовуватиметься для взаємодії з вище описаною клієнтською частиною програмної системи.
10. Згенерувати програмний код реалізації спроектованої API системи згідно спроектованої архітектури. Добавити коментарі та нотації.
11. Підготувати звіт до лабораторної роботи.

За результатами виконаної роботи сформулюйте висновки, в яких зазначте узагальнені рекомендації щодо області застосування заданої у варіанті завдання моделі архітектури.

Теоретичні відомості: Архітектура «клієнт-сервер» є одним із архітектурних шаблонів програмного забезпечення, також це домінуюча в наш час концепція у створенні

розподілених мережних застосунків, яка передбачає взаємодію та обмін даними між ними. Вона передбачає такі компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери повинні бути незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів.

Дуже важливо ясно уявляти, що таке «клієнт». Так можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів — клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

- модель тонкого клієнта, в рамках якої вся логіка застосунку та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;
- модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Для роботи з системою користувач використовує стандартне програмне забезпечення – звичайний браузер. Це позбавляє його необхідності завантажувати спеціальне ПЗ (хоча інколи така необхідність все-таки виникає).

Веб-оглядач формує запит та пересилає його до сервера, який здійснює подальшу обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Висновки

Протягом перших років роботи програмістом ви зазвичай реалізуєте конкретні рішення конкретних проблем. Однак, якщо ви зануритеся глибше в програмування, то помітите, що деякі з цих проблем можна з легкістю виправити на етапі проектування програмного забезпечення, за допомогою вдалого архітектурного рішення. Патерн проектування — це типовий план, який забезпечує загальне вирішення подібних проблем, з якими ви будете стикатися знову і знову у своїй кар'єрі програміста. На відміну від готових функцій чи бібліотек, патерн не можна просто взяти й скопіювати в програму. Патерн являє собою не якийсь конкретний код, а загальний принцип вирішення певної проблеми, який майже завжди треба підлаштовувати для потреб тієї чи іншої програми. Протягом даного курсу ви навчались проводити аналіз предметної області, щоб побачити проблему; правильно обирати відповідні архітектурні моделі; проектувати структуру класів, складових рішення; за допомогою IBM Rational Software Architect автоматизовано генерувати код однією з мов програмування; дописувати вручну особливості реалізації в різних контекстах; закладати в програмі зв'язки з іншими патернами.

Література

1. IBM Featured products [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/products/.
2. Software Deployment Best Practices [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=ZhiCmxGkS-E&list=PL0FD4E604623774DF&index=2.
3. 2/4 - Services Modeling in Rational Software Architect v7.5.4 [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=uYkAzPKtwOM&list=PL0FD4E604623774DF&index=8.
4. DevOps Code Patterns [Електронний ресурс] – Режим доступу до ресурсу: https://developer.ibm.com/devpractices/devops/patterns/.
5. CS50 Style - Shorts-1 (UA) [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=Tmd5C-nyT6M.
6. Занурення в Патерни Проектування. Декоратор [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/decorator.
7. Занурення в Патерни Проектування. Легковаговик [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/flyweight.
8. Занурення в Патерни Проектування. Фасад [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/facade.
9. Занурення в Патерни Проектування. Спостерігач [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/observer.
10. Занурення в Патерни Проектування. Відвідувач [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/visitor.
11. Занурення в Патерни Проектування. Стратегія [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/ strategy.
12. Занурення в Патерни Проектування. Ланцюг відповідальності [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/chain-of-responsibility.
13. Занурення в Патерни Проектування. Стан [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/state.
14. Занурення в Патерни Проектування. Компонувальник [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns/composite.

Навчальний посібник

Лабораторний практикум з розділу «Шаблони проектування» дисципліни «Архітектура та проектування програмного забезпечення»

Упорядники:

М. Петрик,
Д. Михалик,
І. Мудрик,
Ю. Стоянов

Відповідальний за випуск М. Петрик

Видавництво ТНТУ ім. Івана Пулюя

46000, Тернопіль, вул. Руська 56

2016