Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №5

з дисципліни "Технології розроблення програмного забезпечення"

Виконав

студент групи ІА–33:

Китченко Д.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Вступ

Патерни проєктування є фундаментальним інструментом в арсеналі сучасного розробника програмного забезпечення. Вони являють собою перевірені часом, елегантні рішення для типових проблем, що виникають в процесі проєктування архітектури програмних систем. Використання патернів дозволяє створювати більш гнучкі, розширювані та легкі для супроводу додатки.

У даній лабораторній роботі детально розглядається структурний патерн «Adapter», який дозволяє організувати спільну роботу класів з несумісними інтерфейсами.

Мета роботи

Вивчити структуру та призначення шаблону «Adapter» та навчитися застосовувати його на практиці для інтеграції сторонніх або застарілих компонентів у існуючу архітектуру програмної системи без її модифікації.

Теоретичні відомості

Шаблон «Adapter»

Призначення патерну: Шаблон "Adapter" (Адаптер) використовується для адаптації інтерфейсу одного об'єкту до іншого. Наприклад, існує декілька бібліотек для роботи з принтерами, проте кожна має різний інтерфейс (хоча однакові можливості і призначення). Має сенс розробити уніфікований інтерфейс (сканування, асинхронне сканування, двостороннє сканування, потокове сканування і тому подібне), і реалізувати відповідні адаптери для приведення бібліотек до уніфікованого інтерфейсу. Це дозволить в програмі звертатися до загального інтерфейсу, а не приводити різні сценарії роботи залежно від способу реалізації бібліотеки. Адаптери також називаються "wrappers" (обгортками).

Проблема: Ви реалізовуєте аудіо-плеєр, який може програвати аудіо різних форматів. Ви почали аналізувати і бачите що для програвання аудіо із різних форматів краще використовувати різні компоненти. У вас у коді з'являється багато логіки з перевіркою, якщо формат один, то викликаємо такий метод у такого компонента, якщо інший, то викликаємо декілька методів у іншого компонента, і т.д. Логіка роботи з компонентами стає досить складною та заплутаною.

Рішення: Для вирішення цих проблем можна використати патерн Адаптер. Визначаємо загальний інтерфейс IPlayer для програвання музики через компоненти. Далі для кожного компонента робимо свій адаптер. Адаптер має посилання на об'єкт компонента та реалізує інтерфейс викликаючи методи з компонента, який він адаптує. Таким чином, адаптер ніби обгортає специфічний компонент і далі весь клієнтський код буде працювати з адаптерами через інтерфейс IPlayer.

Переваги та недоліки:

+ Відокремлює інтерфейс або код перетворення даних від основної бізнес логіки.

+ Можна добавляти нові адаптери не змінюючи код у класі Client.

- Умовним недоліком можна назвати збільшення кількості класів, але за рахунок використання патерна Адаптер програмний код, як правило, стає легше читати.

Хід роботи

В рамках розробки програми "Архіватор" виникла задача додати підтримку умовного RAR-формату. Для роботи з цим форматом була створена імітація "старої" бібліотеки LegacyRarEngine, API якої є несумісним зі стандартним інтерфейсом IArchiverStrategy, що використовується в системі.

Пряма інтеграція LegacyRarEngine у клас RarStrategy призвела б до значного ускладнення коду та порушення принципу єдиної відповідальності. LegacyRarEngine має низку "незручностей", що роблять його несумісним з існуючою архітектурою:

- Повертає коди помилок (-1, 0) замість сучасних винятків.
- Надає доступ до файлів в архіві тільки за індексом, а не за іменем.
- Використовує застарілі формати даних (наприклад, long timestamp замість LocalDateTime).
- Використовує власну структуру даних RarRecord з публічними полями.

Для вирішення цієї проблеми було спроєктовано багаторівневу архітектуру з використанням патерну "Адаптер":

1. LegacyRarEngine (Adaptee): Компонент, що адаптується. Це наша "стара" бібліотека з несумісним API.
2. RarAdapter (Adapter): Клас-адаптер, який "обгортає" LegacyRarEngine. Він бере на себе всю складність "спілкування" зі старою бібліотекою, перетворюючи коди помилок у винятки, ховаючи логіку доступу за індексами та конвертуючи формати даних.
3. RarStrategy (Client): Клас-клієнт, який реалізує стандартний для системи інтерфейс IArchiverStrategy. Замість того, щоб працювати напряму зі складною бібліотекою, він делегує всі виклики до RarAdapter, який надає йому зручний та сучасний інтерфейс.

Такий підхід дозволив ізолювати несумісний компонент і інтегрувати його в систему, не змінюючи основної архітектури та зберігши її чистоту.

Діаграма класів реалізованої системи

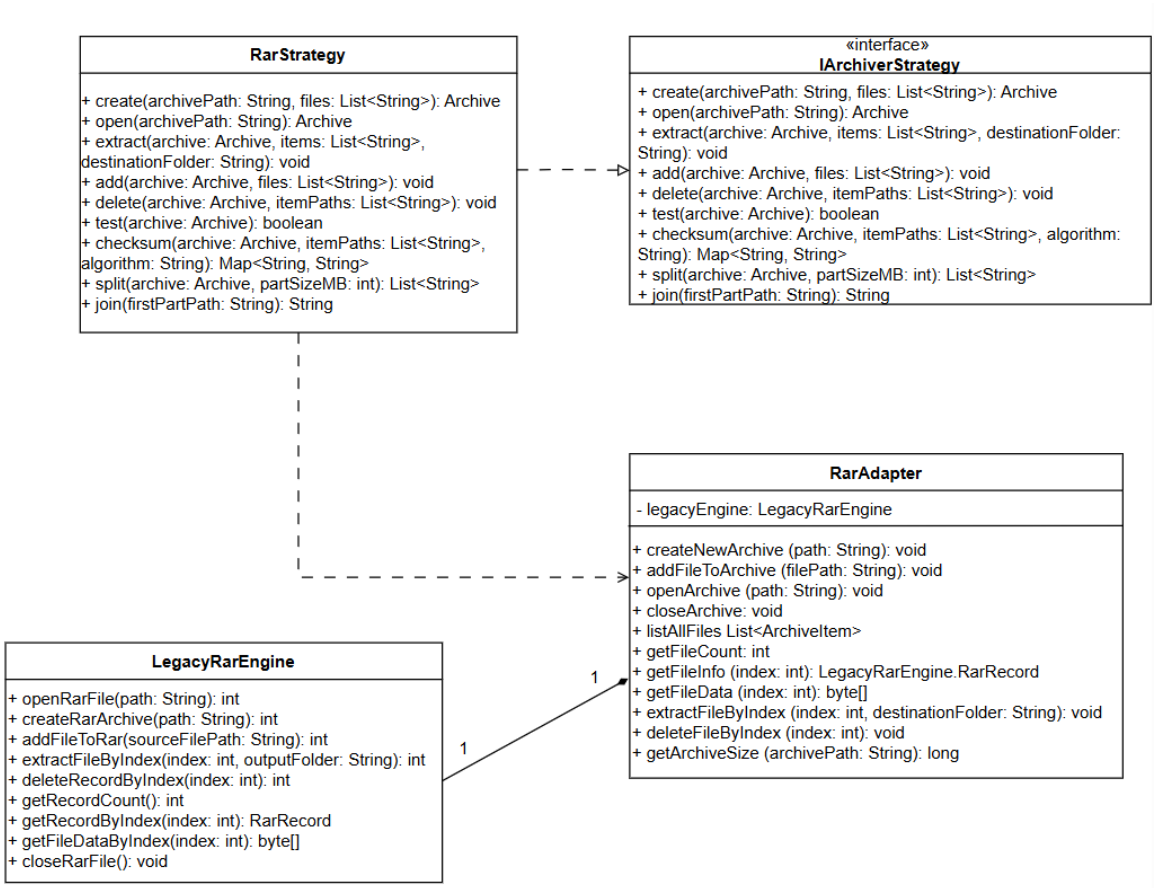


Рисунок 1. Діаграма класів для реалізації патерну Adapter

Фрагменти коду реалізації

1. "Незручний" API класу LegacyRarEngine. Цей клас навмисно спроєктований зі застарілими підходами: повертає коди помилок та надає доступ до файлів за індексом.
2. Логіка адаптації у класі RarAdapter. RarAdapter бере на себе всю складність роботи з LegacyRarEngine, перетворюючи виклики та формати даних.
3. "Чистий" код клієнта RarStrategy. Клас RarStrategy тепер виглядає дуже просто, оскільки він працює тільки зі зручним адаптером.

Тестування реалізації

Для перевірки коректності роботи реалізованої архітектури було створено тестовий клас TestRarAdapter. Цей клас виконує повний цикл операцій: створення архіву, його відкриття, отримання списку файлів, розпакування та тестування.

Результати виконання тесту:

ТЕСТ RAR ADAPTERa

Створення архіву...

RarStrategy: створення RAR архіву test.rar

Створено: test.rar

Розмір архіву: 216

Файли в архіві:

- /
- test_file1.txt
- test_file2.txt

Відкриття архіву...

RarStrategy: відкриття RAR архіву test.rar

Відкрито: 2 файлів

Створюємо папку для витягування...

Витягування файлів...

RarStrategy: витягування з RAR test.rar

Витягнуто: test_file1.txt

Витягнуто: test_file2.txt

Розпакування завершено в extracted

Витягнуто!

Тест цілісності файлів...

RarStrategy: тест RAR архіву test.rar

Тест завершено: OK

ВСЕ ПРАЦЮЄ!

Process finished with exit code 0

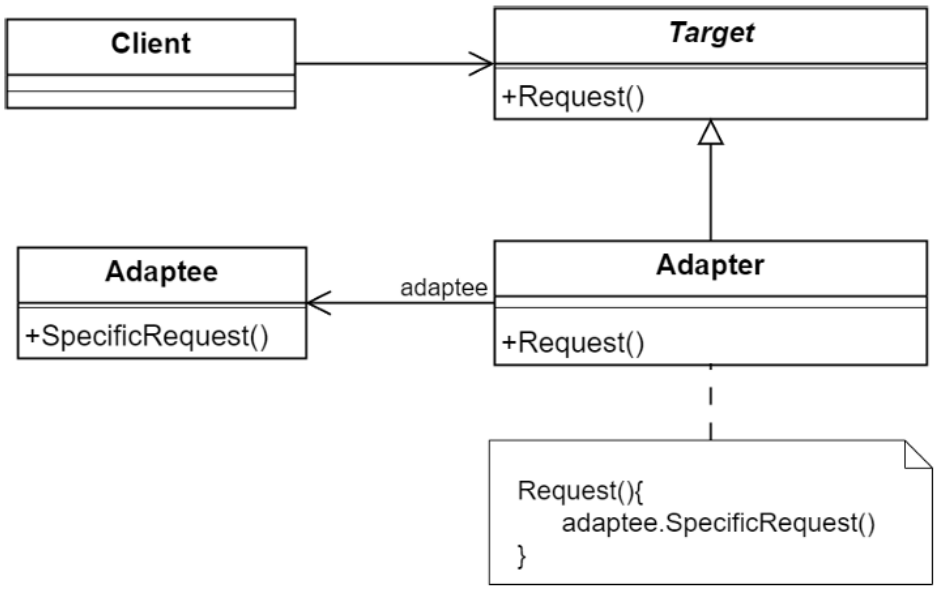
Виконання тестового класу демонструє, що всі етапи роботи з "legacy" архівом через RarStrategy та RarAdapter проходять успішно.

Питання до лабораторної роботи

1. Яке призначення шаблону «Адаптер»?

   Призначення шаблону «Адаптер» - забезпечити спільну роботу класів з несумісними інтерфейсами. Він діє як "перехідник" або "обгортка" навколо існуючого класу (Adaptee), перетворюючи його інтерфейс на інший, очікуваний клієнтським кодом (Client).

2. Нарисуйте структуру шаблону «Адаптер».



3. Які класи входять в шаблон «Адаптер», та яка між ними взаємодія?

   Client (Клієнт): Клас, який хоче використовувати функціонал, але очікує певний інтерфейс (Target). У нашій роботі це RarStrategy.

   Target (Цільовий інтерфейс): Інтерфейс, який використовує Client.

   Adapter (Адаптер): Клас, який реалізує інтерфейс Target і містить посилання на об'єкт Adaptee. Він перенаправляє виклики від Client до Adaptee, виконуючи необхідні перетворення. У нашій роботі це RarAdapter.

   Adaptee (Об'єкт, що адаптується): Існуючий клас з несумісним інтерфейсом. У нашій роботі це LegacyRarEngine.

4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

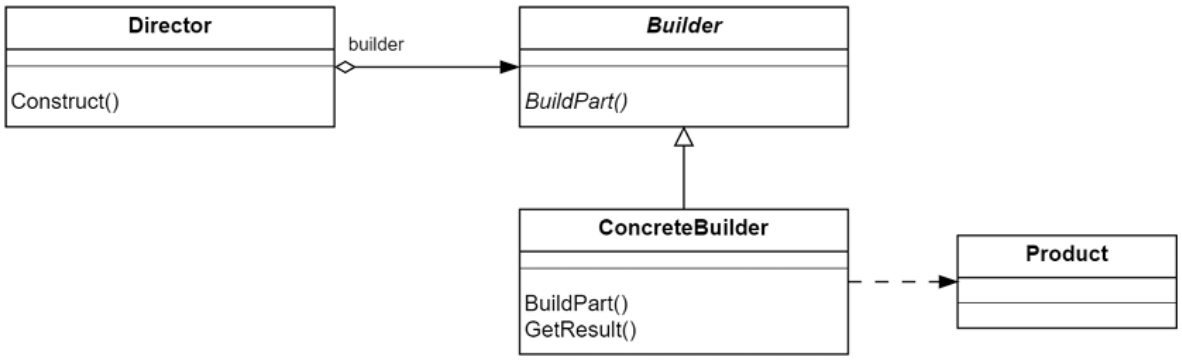
Адаптер об'єктів (використано в роботі): Використовує композицію. Клас Adapter містить екземпляр класу Adaptee. Цей підхід є більш гнучким, оскільки дозволяє адаптувати будь-який підклас Adaptee.

Адаптер класів: Використовує множинне успадкування (в Java реалізується через успадкування класу та реалізацію інтерфейсу). Клас Adapter одночасно успадковує Adaptee та реалізує інтерфейс Target. Цей підхід менш гнучкий.

5. Яке призначення шаблону «Будівельник»?

Шаблон «Будівельник» (Builder) використовується для покрокового створення складних об'єктів. Він дозволяє відокремити процес конструювання об'єкта від його представлення, завдяки чому один і той самий процес конструювання може створювати різні представлення.

6. Нарисуйте структуру шаблону «Будівельник».



7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

Product (Продукт): Складний об'єкт, який створюється.

Builder (Будівельник): Абстрактний інтерфейс для створення частин об'єкта Product.

ConcreteBuilder (Конкретний будівельник): Реалізує інтерфейс Builder і конструює конкретне представлення продукту.

Director (Директор): Клас, який керує процесом побудови, використовуючи об'єкт Builder.

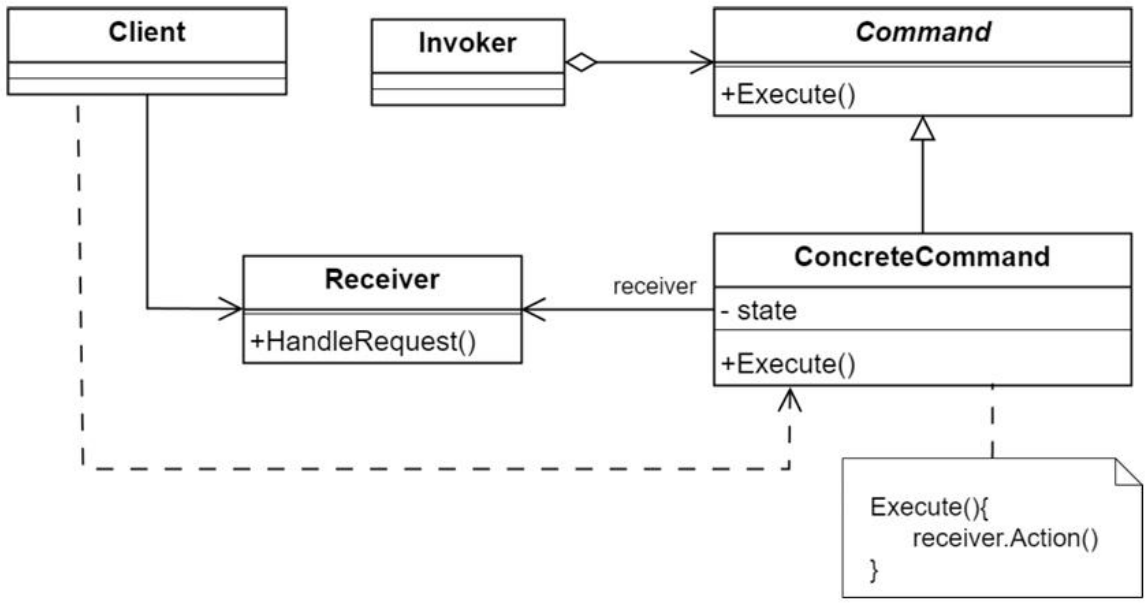
8. У яких випадках варто застосовувати шаблон «Будівельник»"?

Коли процес створення об'єкта є складним, багатоетапним, або коли конструктор має занадто багато параметрів (особливо опціональних). Також, коли потрібно створювати різні представлення одного й того ж об'єкта.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» (Command) інкапсулює запит на виконання дії як об'єкт. Це дозволяє параметризувати клієнтські об'єкти різними запитами, ставити запити в чергу, логувати їх, а також підтримувати операції скасування (undo).

10. Нарисуйте структуру шаблону «Команда».



11. Які класи входять в шаблон «Команда», та яка між ними взаємодія?

Command: Інтерфейс, що оголошує метод для виконання операції (execute).

ConcreteCommand: Реалізує інтерфейс Command, містить посилання на Receiver і викликає його методи.

Client: Створює об'єкт ConcreteCommand і встановлює його одержувача.

Invoker: Просить команду виконати запит.

Receiver: "Одержувач", який знає, як виконати операцію.

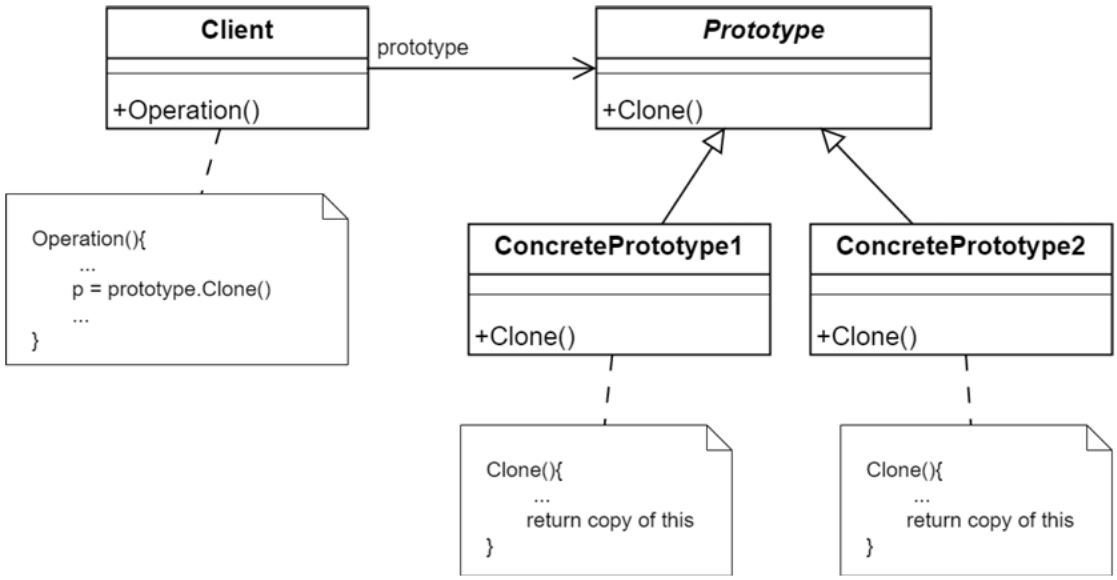
12. Розкажіть як працює шаблон «Команда».

Клієнт створює об'єкт-команду, пов'язуючи її з конкретним одержувачем. Потім цей об'єкт-команда передається ініціатору (Invoker), наприклад, кнопці в меню. Коли користувач натискає кнопку, ініціатор викликає метод execute() у команди, а команда, в свою чергу, викликає потрібний метод у свого одержувача.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» (Prototype) дозволяє створювати нові об'єкти шляхом копіювання існуючого об'єкта (прототипу). Це дозволяє уникнути прив'язки до класів об'єктів, що створюються.

14. Нарисуйте структуру шаблону «Прототип».



15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

Prototype: Інтерфейс, що оголошує метод клонування (clone).

ConcretePrototype: Реалізує інтерфейс Prototype та метод clone.

Client: Створює новий об'єкт, викликаючи метод clone у прототипу.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Обробка подій в GUI: Коли користувач клікає на кнопку, подія спочатку обробляється кнопкою, потім може бути передана батьківській панелі, потім вікну, і так далі, доки не буде оброблена.

Системи логування: Повідомлення може проходити через ланцюжок обробників: один записує в консоль, інший - у файл, третій - відправляє по email, залежно від рівня важливості.

Системи авторизації та валідації: Запит користувача може проходити через ланцюжок перевірок: перевірка автентифікації, перевірка прав доступу, перевірка валідності даних.

Висновки

В ході виконання даної лабораторної роботи було В ході виконання даної лабораторної роботи було детально вивчено та успішно застосовано на практиці структурний патерн проєктування «Adapter».

Було вирішено практичну задачу інтеграції стороннього компонента (LegacyRarEngine) з несумісним API в існуючу архітектуру програми "Архіватор". Створення класу RarAdapter дозволило повністю інкапсулювати складність роботи зі "старою" бібліотекою, включаючи перетворення кодів помилок у винятки, адаптацію форматів даних та приховування незручної логіки доступу.

Інтеграція була проведена в рамках гнучкої архітектури, що базується на патерні «Strategy», який дозволяє динамічно обирати алгоритм роботи з архівами різних форматів (таких як ZIP, RAR та ін.). Патерн «Adapter» органічно доповнив цю архітектуру, дозволивши додати підтримку нового формату без модифікації існуючих клієнтських класів чи основної бізнес-логіки