Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №5

Технологія розробки програмного забезпечення

Шаблони «Adapter», «Builder», «Command», «Chain of Responsobility», «Prototype»

Виконала студентка
групи ІА-23:
Кашуб’як С. М.

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Шаблони «Adapter», «Builder», «Command», «Chain of Responsobility», «Prototype»

Мета: Ознайомитися з принципами роботи шаблонів проектування «Adapter», «Builder», «Command», «Chain of Responsobility», «Prototype», їх перевагами та недоліками. Набути практичних навичок у застосуванні шаблону adapter при розробці програмного забезпечення на прикладі реалізації архіватора.

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми

Варіант:

..14 Архіватор (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

ЗМІСТ

Теоретичні відомості

Анти-шаблони (анти-патерни) – це поширені рішення, які здаються правильними на перший погляд, але насправді вони можуть призвести до проблем у проектуванні і розвитку програмного забезпечення. Вони часто виникають через недостатнє розуміння проблеми або через використання швидких рішень, що не враховують довгострокові наслідки. Замість того, щоб бути кращими практиками, анти-шаблони можуть стати джерелом технічного боргу, що ускладнює підтримку і розширення систем

Шаблон адаптер (Adapter) є структурним шаблоном проектування, який дозволяє змінювати інтерфейс одного класу так, щоб він став сумісним з інтерфейсом іншого класу. Це корисно, коли вам потрібно інтегрувати компоненти або системи, які мають різні інтерфейси, але ви не хочете змінювати їх код.

Як працює адаптер: адаптер має на меті забезпечити перехід між двома несумісними інтерфейсами. Він створює клас, який "адаптує" один інтерфейс до іншого. Ключова ідея - це не змінювати класи, що мають різні інтерфейси, а створити додатковий клас, який допоможе їм працювати разом.

Структура:

- Target (Цільовий інтерфейс) – це інтерфейс, який очікує клієнт.
- Client (Клієнт) – це клас, який використовує Target.
- Adaptee (Адаптований клас) – це клас з існуючим інтерфейсом, який потрібно адаптувати.
- Adapter (Адаптер) – це клас, що реалізує Target і делегує виклики Adaptee.

Переваги:

- Збереження існуючого коду – адаптер дозволяє використовувати старі класи без необхідності їх зміни.
- Гнучкість – легко додаються нові адаптери для підтримки різних інтерфейсів.
- Ізоляція змін – клієнт не потребує знань про зміни в адаптованому класі, що спрощує його використання.

Недоліки:

- Збільшення складності – додавання додаткових класів може ускладнити код, особливо якщо адаптерів багато.
- Перевантаження – кожен адаптер вимагає додаткової роботи для налаштування, що може вплинути на продуктивність.

Хід Роботи

Рис 1. Діаграма класів

Діаграма класів

1. Інтерфейс ArchiveAdapter (Target)

   Роль – абстрактний контракт, який визначає набір операцій для роботи з архівами.

   Методи:

   - createArchive(String archiveName, String[] files): створення архіву із заданими файлами.
   - extractArchive(String archiveName, String outputDir): розпакування архіву у вказану директорію.
   - addFile(String archiveName, String filePath): додавання файлу до архіву.
   - deleteFile(String archiveName, String fileName): видалення файлу з архіву.

   Призначення: Уніфікація інтерфейсу для роботи з архівами різних форматів. Це дозволяє клієнту (код у класі Main) працювати з будь-яким архівом через спільний інтерфейс, не залежачи від специфічної реалізації.

2. Класи-адаптери

   Кожен із цих класів реалізує інтерфейс ArchiveAdapter і адаптує специфічні методи для конкретного типу архіву.

   ZipArchiveAdapter

   Роль – адаптер для роботи з архівами формату .zip.

   Методи:

- createArchive: викликає метод для створення архіву .zip.
- extractArchive: реалізує логіку розпакування .zip архіву.
- addFile: додає файл у .zip архів.
- deleteFile: видаляє файл із .zip архіву.

Призначення: Дозволяє працювати з архівами формату .zip через уніфікований інтерфейс.

TarGzArchiveAdapter

Роль – адаптер для роботи з архівами формату .tar.gz.

Методи: аналогічні до ZipArchiveAdapter, але реалізовані для формату .tar.gz.

RarArchiveAdapter

Роль – адаптер для роботи з архівами формату .rar.

Методи: реалізує специфічну логіку для роботи з .rar.

AceArchiveAdapter

Роль – адаптер для роботи з архівами формату .ace.

Методи: реалізує специфічну логіку для роботи з .ace.

3. Взаємодія класів
   - Спільна основа – кожен адаптер реалізує інтерфейс ArchiveAdapter. Це дозволяє клієнтському коду (наприклад, у класі Main) працювати з архівами будь-якого типу, не знаючи деталей їх реалізації.
   - Клієнт (Main) – використовує адаптери через інтерфейс ArchiveAdapter, викликаючи однакові методи для різних форматів архівів.

4. Механізм розширення
   - Додавання нових форматів – щоб додати підтримку нового типу архіву, потрібно створити новий адаптер (наприклад, SevenZipArchiveAdapter), що реалізує інтерфейс ArchiveAdapter.
   - Це дозволяє зберегти гнучкість системи та уникнути модифікації існуючого коду, дотримуючись принципу відкритості/закритості (OCP) з SOLID.

Рис 2. Застосування шаблону при реалізації програми

Робота патерну

Створення архіву

Опис дії – створюється архів archive.zip, який містить список файлів.

Реалізація: вибирається конкретний адаптер (ZipArchiveAdapter) для роботи з .zip-архівами. Метод createArchive викликається через інтерфейс ArchiveAdapter. Він делегує створення архіву специфічній реалізації в ZipArchiveAdapter.

Роль патерну Adapter – у цьому кроці використовується уніфікований інтерфейс для створення архіву. Завдяки Adapter, клієнтський код не знає, який саме тип архіву створюється.

Розпакування архіву

Опис дії – архів archive.zip розпаковується в директорію ./output.

Реалізація: метод extractArchive викликається через інтерфейс ArchiveAdapter. Логіка розпакування архіву реалізована в ZipArchiveAdapter.

Роль патерну Adapter: уніфікація процесу розпакування для різних типів архівів. Клієнтський код не залежить від формату архіву.

Додавання файлу до архіву

Опис дії – до архіву archive.zip додається файл example.txt.

Реалізація: викликається метод addFile через інтерфейс ArchiveAdapter Специфічна логіка додавання файлу реалізована в ZipArchiveAdapter.

Роль патерну Adapter: адаптер уніфікує роботу з різними архівами. Наприклад, для .rar або .tar.gz реалізація може відрізнятися, але клієнтський код залишається незмінним.

Видалення файлу з архіву

Опис дії – з архіву archive.zip видаляється файл example2.txt.

Реалізація: викликається метод deleteFile через інтерфейс ArchiveAdapter. Логіка видалення файлу реалізована в ZipArchiveAdapter.

Роль патерну Adapter: забезпечується спільний інтерфейс для операції видалення файлу з архіву.

Переваги використання шаблону Adapter у цьому випадку

Роль Adapter – усі операції з архівами виконуються через уніфікований інтерфейс ArchiveAdapter, що забезпечує прозору роботу з різними форматами архівів.
Переваги:

- Спрощення клієнтського коду.
- Легкість у додаванні підтримки нових форматів архівів (наприклад, .ace, .rar) без зміни клієнтського коду.
- Механізм: вибір адаптера (ZipArchiveAdapter, TarGzArchiveAdapter, тощо) відбувається під час виконання, але клієнтському коду це невідомо, що є основним принципом шаблону Adapter.

Висновок: отже, у ході виконання лабораторної роботи було реалізовано шаблон проектування Adapter, що дозволяє уніфікувати взаємодію з різними типами архівів, такими як .zip, .rar, .tar.gz та .ace. Завдяки використанню цього шаблону вдалося створити гнучку і масштабовану архітектуру, де клієнтський код взаємодіє з архівами через єдиний інтерфейс, а специфічна логіка для кожного формату архіву прихована в адаптерах.

Посилання на код: https://github.com/SofiiaKashubiak/Archivator-project.git