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:
Шрубович Н. С.

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:

..3 Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

ЗМІСТ

Теоретичні відомості 4
Хід Роботи 5
Діаграма класів 5
Робота патерну 7
Переваги використання шаблону Command 8
Висновок 8
Посилання на код 8

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 команда (Command) 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 шаблон "Команда"

Шаблон дозволяє інкапсулювати виклики методів як об’єкти. Замість того, щоб напряму викликати методи об’єкта, ви створюєте об'єкт-команду, який виконує запит. Цей підхід дозволяє реалізувати такі функції: Виконання команд у певний час. Скасування та повтор команд. Збереження історії виконаних команд.

Структура

Command (Команда) – це інтерфейс, що визначає метод execute() для виконання запиту.
ConcreteCommand (Конкретна команда) – реалізує інтерфейс Command та інкапсулює запит до Receiver.
Receiver (Одержувач) – виконує фактичну операцію, яку інкапсулює команда.
Invoker (Ініціатор) – відповідає за зберігання та виклик об’єктів команд.
Client (Клієнт) – створює конкретні команди та передає їх ініціатору.

Приклад роботи шаблону "Команда"

Уявіть, що ви працюєте над текстовим редактором. Користувач може виконувати операції "Зберегти", "Видалити" або "Скасувати". Кожна з цих дій буде інкапсульована в окремий об'єкт команди.

Переваги шаблону "Команда":

- Ізоляція логіки: Клієнт створює і передає команди, не знаючи їх реалізації.
- Гнучкість: Нові команди додаються без зміни існуючого коду.
- Скасування і повтор: Легко реалізується "undo/redo".
- Композиція: Команди комбінуються для складних сценаріїв.

Недоліки шаблону "Команда":

- Складність: Велика кількість класів ускладнює систему.
- Ресурси: Зберігання історії команд потребує додаткової пам’яті.

Хід Роботи

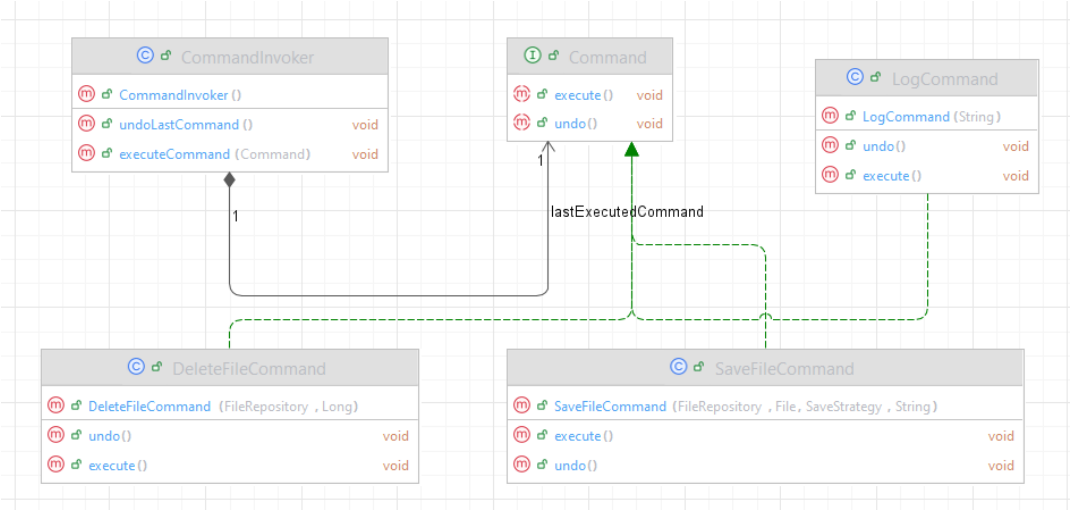

Рис 1. Діаграма класів

Діаграма класів

1. Інтерфейс Command

   Роль- визначає абстрактний контракт для виконання операцій, які будуть інкапсульовані у вигляді команд.

   Методи:

   execute(): виконує основну операцію.

   undo(): скасовує виконану операцію.

   Призначення:

   Уніфікація підходу до виконання дій через команди, забезпечуючи ізоляцію клієнта від конкретної реалізації.

2. Класи команд

Кожен із цих класів реалізує інтерфейс Command, інкапсулює специфічні операції та делегує їх одержувачу.

- SaveFileCommand:

  Роль: Реалізує збереження файлу за допомогою вибраної стратегії.

  Методи:

  - execute: викликає метод збереження файлу через стратегію.
  - undo: видаляє збережений файл із репозиторію.

    Призначення: Забезпечує збереження файлів незалежно від їх типу.
- DeleteFileCommand:

  Роль: Реалізує видалення файлу з репозиторію.

  Методи:

- execute: видаляє файл.
- undo: (опціонально) повертає файл у репозиторій.
  Призначення: Забезпечує безпечне видалення з можливістю скасування.

- LogCommand:
  Роль: Логування операцій.
  Методи:
  - execute: записує повідомлення у лог.
  - undo: немає застосування.
    Призначення: Відстеження виконаних операцій.

- Клас CommandInvoker (Invoker)
  Роль:Виконує збереження та виклик об’єктів команд.

  Методи:

  - executeCommand(Command command): викликає метод execute() у переданого об'єкта команди.
  - undoLastCommand(): викликає метод undo() у останньої виконаної команди.
  - Призначення:Забезпечує механізм виконання та скасування операцій без потреби знати, як вони реалізовані.

3. Взаємодія класів
   Client створює об'єкти конкретних команд (ConcreteCommand) та передає їх у CommandInvoker.
   CommandInvoker викликає метод execute() у команд, які делегують операції Receiver.
   Усі команди реалізують інтерфейс Command, що забезпечує єдиний підхід до роботи з ними.

4. Механізм розширення
   Додавання нових команд: Для підтримки нових операцій достатньо створити новий клас, що реалізує інтерфейс Command.
   Додавання нових стратегій: Стратегії збереження файлів (SaveStrategy) легко розширюються через створення нових реалізацій, таких як YamlSaveStrategy. Система залишається гнучкою та легко розширюється, дотримуючись принципу відкритості/закритості (OCP) з SOLID.

Робота патерну

Патерн "Команда" у системі інкапсулює операції як окремі об’єкти, що дозволяє реалізувати їх виконання, скасування та повторення без прив’язки до конкретної логіки. Основними учасниками є інтерфейс Command, класи команд (SaveFileCommand, DeleteFileCommand, LogCommand), ініціатор (CommandInvoker), одержувачі (FileRepository, стратегії збереження), а також клієнтський код (наприклад, клас Main).

Робота патерну починається з клієнта, який створює конкретну команду. Наприклад, у випадку збереження файлу клієнт створює об’єкт SaveFileCommand, передаючи у нього необхідні параметри: файл, репозиторій файлів та обрану стратегію збереження. Потім команда передається ініціатору — об’єкту CommandInvoker, який відповідає за виконання операції.

Коли CommandInvoker викликає метод execute() команди, остання делегує виконання операції одержувачу. У випадку SaveFileCommand це може бути виклик стратегії збереження (SaveStrategy), яка реалізує специфічну логіку для збереження файлу у форматах TXT, JSON або XML. Таким чином, основна робота збереження інкапсулюється в окремих класах, а клієнт не потребує знань про внутрішню реалізацію.

У системі реалізована підтримка скасування дій. Для цього CommandInvoker зберігає історію виконаних команд у стеку. Якщо клієнт викликає метод undoLastCommand(), CommandInvoker звертається до останньої команди у стеку та викликає її метод undo(). Наприклад, для команди SaveFileCommand це може бути видалення файлу з репозиторію.

Логування операцій реалізовано через команду LogCommand. Вона додає записи до логу, наприклад, після успішного виконання операції збереження або видалення файлу. Це дозволяє відстежувати всі виконані дії.

Патерн також підтримує розширення функціональності. Наприклад, для додавання операції "перейменування файлу" достатньо створити нову команду RenameFileCommand, яка реалізує інтерфейс Command і виконує логіку перейменування. Клієнтський код при цьому залишається незмінним, що забезпечує гнучкість і масштабованість системи.

Таким чином, патерн "Команда" дозволяє ізолювати клієнтський код від деталей реалізації операцій, інкапсулювати логіку виконання та забезпечувати підтримку додаткових функцій, таких як скасування або логування, без порушення структури системи.

Переваги використання шаблону Command у цьому випадку

Шаблон "Команда" у цьому випадку забезпечує ізоляцію клієнтського коду від деталей реалізації операцій, що робить систему більш гнучкою і масштабованою. Завдяки інкапсуляції операцій у вигляді команд можна легко додавати нові функції без зміни існуючого коду, дотримуючись принципу відкритості/закритості. Крім того, патерн дозволяє реалізувати механізми скасування та повтору дій, що підвищує зручність використання системи.

Висновок: Отже, у ході виконання лабораторної роботи було реалізовано шаблон проектування Command, що дозволяє інкапсулювати операції у вигляді об'єктів команд. Завдяки використанню цього шаблону вдалося забезпечити ізоляцію клієнтського коду від деталей реалізації операцій, підтримати механізми скасування та повтору дій, а також створити гнучку і масштабовану архітектуру. Це дозволяє легко додавати нові функціональні можливості без змін у існуючому коді, забезпечуючи простоту підтримки та розширення системи.

Посилання на код: https://github.com/Nata198111/TRPZ/tree/master