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

Технології розроблення програмного забезпечення

Тема: «Патерни проектування.»

Виконала:

Студентка групи ІА-34

Сизоненко А.О

Перевірив:

Мягкий Михайло Юрійович

Київ-2025

Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

Тема роботи: 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).

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 «Adapter» використовується для приведення інтерфейсу одного об’єкта до інтерфейсу іншого, з яким працює програма. Це корисно, коли потрібно інтегрувати компоненти або бібліотеки з різними інтерфейсами, але однаковим функціоналом, не змінюючи їх внутрішній код. Адаптери дозволяють створити єдиний уніфікований інтерфейс для роботи з різними реалізаціями.

Шаблон «Builder» використовується для поетапного створення складних об’єктів, відокремлюючи процес створення від представлення. Це дозволяє легко створювати різні варіанти одного об’єкта або об’єкти, які потребують багато налаштувань, без нагромадження конструкторів із великою кількістю параметрів.

Шаблон «Chain of Responsibility» організовує обробку запиту у вигляді ланцюга об’єктів, де кожен об’єкт має можливість обробити запит або передати його далі. Це дозволяє розділити обов’язки між кількома об’єктами і спростити логіку прийняття рішень, наприклад, при підписанні документів або обробці подій.

Шаблон «Prototype» дозволяє створювати нові об’єкти шляхом клонування вже існуючих прототипів. Це зручно, коли відомо, як має виглядати кінцевий об’єкт, або коли створення об’єкта з нуля є складним або ресурсозатратним. Використання прототипів зменшує ієрархію класів і спрощує створення складних об’єктів під час виконання програми.

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

1. Діаграма класів для реалізації шаблону команда.

Рисунок 1 – Діаграма класів

Опис діаграми класів

1. FileRepository – в даному випадку виступає в ролі receirver, тобто має реальну реалізацію для функцій save та delete.
2. CommandInvoker — це Invoker. Він керує виконанням команд і веде історію (стек) для Undo. Він просто викликає execute() і undo().
3. Command - це інтерфейс, який задає загальні методи execute() та undo(). Він інкапсулює дію, дозволяє працювати з різними командами однаковим чином.
4. Client — ініціатор. Створює об’єкти (FileRepository, CommandInvoker, файли), створює команди (SaveFileCommand, DeleteFileCommand) і передає їх Invoker для виконання. Client також керує порядком виконання та Undo команд.
5. SaveFileCommand та DeleteFileCommand — це ConcreteCommand(класи конкретних команд). Вони реалізують інтерфейс Command у execute() вони викликають методи Receiver, а в undo() роблять протилежну операцію (для Save — delete, для Delete — save).

6. InsertCharCommand – це клас конкретної команди, що реалізує ітерфейс Command у execute() вставляє символ у файл на конкретну позицію та в undo() відповідно його видаляє.

Приклад реалізації шаблону:

Спочатку Client створює об’єкти, які беруть участь у патерні:

- Створюється FileRepository — Receiver.
- Створюється CommandInvoker — Invoker, який керує виконанням команд і зберігає історію для Undo.
- Створюється об’єкт File, що містить інформацію про файл (назву, шлях, контент).

Далі Client створює конкретні команди:

- SaveFileCommand — команда для збереження файлу.
- DeleteFileCommand — команда для видалення файлу.
- InsertCharCommand — команда для вставки символу у файл.

Потім Client передає команди Invoker для виконання:

1. Invoker викликає execute() на SaveFileCommand. Команда делегує роботу FileRepository, і файл реально зберігається на диску або у потрібному форматі.
2. Invoker викликає execute() на DeleteFileCommand. Команда делегує FileRepository, який видаляє файл.
3. Invoker викликає execute() на InsertCharCommand. Команда вставляє символ у контент файлу і зберігає зміни через FileRepository.

Якщо потрібно відмінити дію:

- Invoker бере останню виконану команду зі стеку і викликає її undo().
- SaveFileCommand при Undo видаляє файл, який раніше зберігся.
- DeleteFileCommand при Undo зберігає файл, який був видалений.
- InsertCharCommand при Undo видаляє вставлений символ і повертає старий стан контенту.

У такій реалізації патерн Command дозволяє Client керувати файлами через команди, не знаючи деталей їх виконання, а Receiver (FileRepository) реалізує саму роботу. Invoker забезпечує збереження історії і можливість Undo.

Демонстрація роботи шаблону Command

```java
public class Client {
    public static void main(String[] args) {
        FileRepository repository = new FileRepository( jsonFilePath: "");
        CommandInvoker invoker = new CommandInvoker();

        File fileJson = new File();
        fileJson.setId(1L);
        fileJson.setFileName("file_json1");
        fileJson.setFilePath("C:\\Users\\АНЮТА\\IdeaProjects\\trpz\\json");
        fileJson.setContent("{ \"text\": \"Hello JSON\" }");

        File fileTxt = new File();
        fileTxt.setId(2L);
        fileTxt.setFileName("file_txt");
        fileTxt.setFilePath("C:\\Users\\АНЮТА\\IdeaProjects\\trpz\\txt");
        fileTxt.setContent("{ \"text\": \"afqkdfjhdf\" }");


        Command save1 = new SaveFileCommand(repository, fileJson);
        Command save2 = new SaveFileCommand(repository, fileTxt);
        Command delete = new DeleteFileCommand(repository,  fileId: 1L);
        InsertCharCommand insertChar = new InsertCharCommand(repository,  fileId: 2L,  character: '>',  position: 5);

        invoker.executeCommand(save1);
        invoker.executeCommand(save2);
        invoker.executeCommand(insertChar);
        invoker.executeCommand(delete);
        invoker.undoLastCommand();
    }
```

Рисунок 2 – код класу Client

Результат виконання програми:

```
Файл збережено як JSON: C:\Users\АНЮТА\IdeaProjects\trpz\json\file_json1.json
File saved: file_json1
Збережено як TXT: C:\Users\АНЮТА\IdeaProjects\trpz\txt\file_txt.txt
File saved: file_txt
Збережено як TXT: C:\Users\АНЮТА\IdeaProjects\trpz\txt\file_txt.txt
Char `>` inserted in File: file_txt
File deleted: file_json1
Файл збережено як JSON: C:\Users\АНЮТА\IdeaProjects\trpz\json\file_json1.json
Undo delete: file_json1
```

Рисунок 3 – вивід повідомлення в консолі

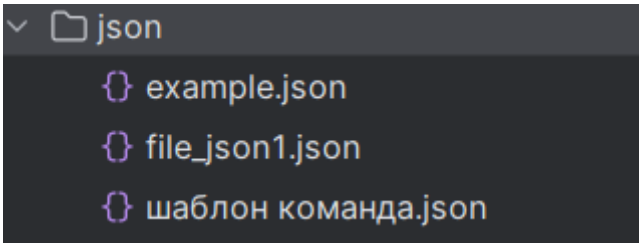

Рисунок 4 – файл збережений у форматі json

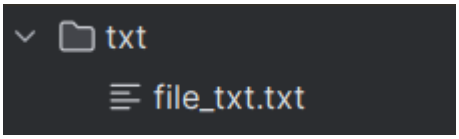

Рисунок 5 – файл збережений у форматі txt

```
"te>xt": "afqkdfjhdf"
```

Рисунок 6 – додавання символу “>” у файл

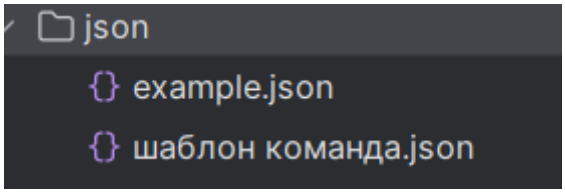

Рисунок 7 – видалення файлу file_json1

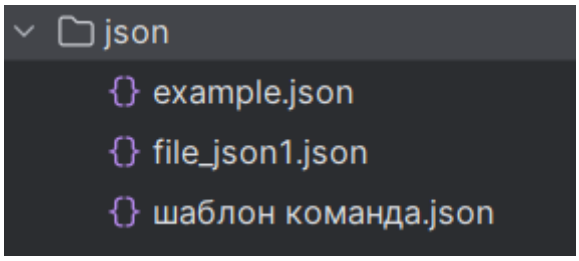

Рисунок 8 – виконання команди undo(збереження файлу file_json1 з пам’яті)

Переваги використання шаблону Command для моєї теми проекту:

1. Можливість скасування дій (Undo)
   Однією з ключових особливостей текстових редакторів є функція "Відмінити". Патерн Command забезпечує простий спосіб реалізувати Undo, оскільки кожна команда зберігає інформацію про те, що вона зробила, і може виконати протилежну дію (наприклад, видалення файлу після збереження або відновлення після видалення).
2. Розширюваність системи
   Коли в майбутньому потрібно буде додати нові можливості (наприклад, команду “Rename”, “Copy”, “Move”), не потрібно змінювати існуючий код. Достатньо створити новий клас команди, що реалізує той самий інтерфейс Command. Це робить проєкт гнучким і легко підтримуваним.
3. Єдине уніфіковане представлення для різних дій
   Усі команди реалізують спільний інтерфейс Command із методами execute() і undo(). Це дозволяє Invoker-у працювати з будь-якою командою однаково, незалежно від того, чи це збереження, видалення чи оновлення файлу.

Недоліки використання шаблону Command у текстовому редакторі:

1. Ускладнення структури коду — для кожної операції (збереження, видалення, перейменування) потрібно створювати окремий клас-команду, що робить проєкт об’ємнішим.
2. Складність реалізації відміни дій — необхідність ретельно продумувати логіку скасування дій (особливо для роботи з файлами).
3. Збільшення використання пам’яті — оскільки зберігається історія виконаних команд для можливості їх відміни, це може сповільнити роботу програми при тривалій сесії.

Висновок: У роботі реалізовано патерн Command для виконання операцій над файлами у текстовому редакторі, таких як збереження, видалення та відміна дій (Undo). Це дозволяє відокремити об’єкт, який ініціює дію, від об’єкта, який її виконує (FileRepository), і забезпечує гнучке керування операціями через єдиний інтерфейс. Основною перевагою є можливість легко додавати нові команди та реалізовувати скасування виконаних дій, а недоліком — збільшення кількості класів і необхідність зберігання історії команд для Undo. Було створено діаграму класів, яка наочно демонструє взаємодію компонентів системи: інтерфейс Command визначає методи execute та undo, конкретні команди (SaveFileCommand, DeleteFileCommand) реалізують ці дії, а клас CommandInvoker керує їхнім виконанням та зберігає історію команд.

Відповіді на контрольні питання

1. Яке призначення шаблону «Адаптер»?
Шаблон «Adapter» (Адаптер) використовується для приведення інтерфейсу одного об’єкта до інтерфейсу іншого, з яким працює програма, без зміни внутрішнього коду об’єкта.

2. Нарисуйте структуру шаблону «Адаптер».
Структура:
Клієнт → Адаптер → Адаптований клас.

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

- Client — використовує інтерфейс адаптера.
- Target — визначає інтерфейс, з яким працює клієнт.
- Adapter — реалізує інтерфейс Target і делегує виклики Adaptee.
- Adaptee — клас з іншим інтерфейсом, який треба адаптувати.

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

Об’єктний адаптер — використовує композицію, Adapter містить посилання на Adaptee.

Класовий адаптер — використовує наслідування, Adapter наслідує Adaptee та реалізує Target.

5. Яке призначення шаблону «Будівельник»?
Шаблон «Builder» (Будівельник) використовується для поетапного створення складних об’єктів і відокремлює процес створення від представлення об’єкта.

6. Нарисуйте структуру шаблону «Будівельник».
Структура:
Director → Builder → ConcreteBuilder → Product

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

- Director — керує процесом побудови.
- Builder — визначає інтерфейс побудови частин об’єкта.
- ConcreteBuilder — реалізує побудову конкретного продукту.
- Product — кінцевий об’єкт, що створюється.

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

- Коли об’єкт має складний процес створення.
- Коли потрібні різні варіанти представлення одного об’єкта.

9. Яке призначення шаблону «Команда»?
Шаблон «Command» використовується для інкапсуляції дії у вигляді окремого об’єкта, що дозволяє відокремити ініціатора дії від виконавця і реалізувати Undo/Redo, логування або черги команд.

10. Нарисуйте структуру шаблону «Команда».
Структура:
Client → Invoker → Command → Receiver

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

- Command (інтерфейс) — визначає методи execute() і undo().
- ConcreteCommand — реалізує команду, делегуючи роботу Receiver.
- Receiver — виконує фактичну дію.
- Invoker — зберігає команду та викликає execute().
- Client — створює команду і передає її Invoker.

12. Розкажіть як працює шаблон «Команда».
Клієнт створює команду і передає її Invoker, який виконує команду через метод execute(). Receiver виконує конкретну операцію. Якщо потрібна відміна дії, Invoker викликає метод undo() конкретної команди.

13. Яке призначення шаблону «Прототип»?
Шаблон «Prototype» дозволяє створювати нові об’єкти шляхом копіювання існуючих прототипів, що спрощує створення складних об’єктів і зменшує ієрархію класів.

14. Нарисуйте структуру шаблону «Прототип».
Структура:
Prototype → ConcretePrototype → Client

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

- Prototype — визначає метод clone().
- ConcretePrototype — реалізує клонування конкретного об’єкта.
- Client — створює нові об’єкти шляхом клонування прототипу.

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

- Обробка запитів на підпис документів через ланцюг співробітників.
- Фільтрація і обробка подій у GUI, де подія проходить через кілька обробників.
- Серверна обробка запитів, де кожен об’єкт ланцюга приймає або передає запит далі.