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

Тема: Патерни проектування.

Виконала
студентка групи ІА–32:
Слюсарева А.А.

Київ 2025

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 5

Тема: Патерни проектування.

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

5.1. Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати один з розглянутих шаблонів за обраною темою.

• Реалізувати не менше 3-х класів відповідно до обраної теми.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

5.2. Теоретичні відомості

Шаблон «Command»

Призначення патерну: Шаблон "command" (команда) перетворить звичайний виклик методу в клас. Таким чином дії в системі стають повноправними об'єктами. Це зручно в наступних випадках:

• Коли потрібна розвинена система команд – відомо, що команди будуть добавлятися;

• Коли потрібна гнучка система команд – коли з'являється необхідність додавати командам можливість відміни, логування і інш.;

• Коли потрібна можливість складання ланцюжків команд або виклику команд в певний час.

Об'єкт команда сама по собі не виконує ніяких фактичних дій окрім перенаправлення запиту одержувачеві (тобто команди все ж виконуються одержувачем), однак ці об'єкти можуть зберігати дані для підтримки додаткових функцій відміни, логування і інш. Наприклад, команда вставки символу може запам'ятовувати символ, і при виклику відміни викликати відповідну функцію витирання символу. Можна також визначити параметр «застосовності» команди (наприклад, на картинці писати не можна) – і використати цей атрибут для засвічування піктограми в меню. Такий підхід до команд дозволяє побудувати дуже гнучку систему команд, що настроюється. У більшості додатків це буде зайвим (використовується спрощений варіант), проте життєво важливий в додатках з великою кількістю команд (редактори).

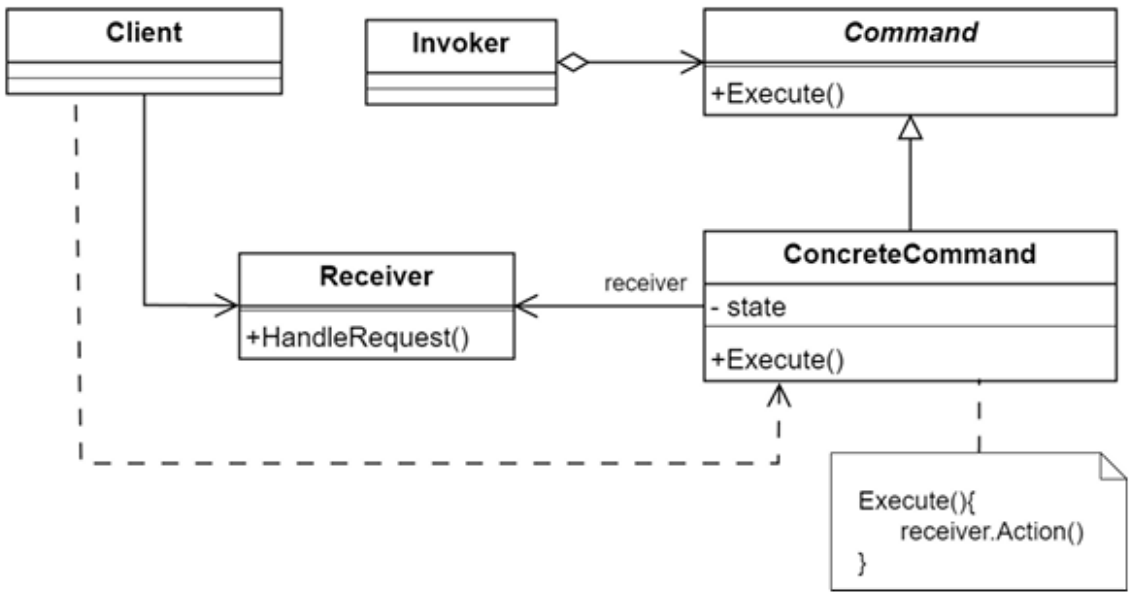


Рисунок 5.3. Структура патерну Команда

Проблема: Ви реалізуєте товстий клієнт який має багатий візуальний інтерфейс: має меню, кнопки і контекстне меню. Кожна дія, яку можна виконати, має три варіанти виконання – через меню, натисканням кнопки та через контекстне меню. Реалізацію кожної дії можна розмістити в обробнику 62 візуального елементу, але тоді потрібно буде продублювати цей функціонал для меню, кнопки та контекстного меню. Таким чином ми будемо мати дублювання коду і при зміні функціоналу змінювати його в трьох місцях.

Рішення: Виділення функціоналу який виконується по натисканню на кнопку в окремий клас дозволяє відвязати візуальну частину від логіки обробки. Таким чином ми будемо мати шар з UI елементами і шар з логікою обробки дій виконаних на UI. Це дозволяє один і той самий об'єкт з шару логіки обробки дій використати для реакцію на натискання кнопки і пункту меню і контекстного меню. Кнопки тепер не знають про конкретні класи реалізації команд, а взаємодіють з об'єктами команд через загальний інтерфейс. Ще одна перевага, що тепер ми можемо достатньо просто реалізувати механізм enable-disable для кнопок та контекстного меню через виклик у об'єкта команди метода IsEnabled() або через прив'язку (binding) на поле IsEnabled у об'єкта команди.

Переваги та недоліки:

+ Ініціатор виконання команди не знає деталей реалізації виконавця команди.

+ Підтримує операції скасування та повторення команд.

+ Послідовність команд можна логувати і при необхідності виконати цю послідовність ще раз.

+ Простота розширення за рахунок додавання нових команд без необхідності внесення змін в уже існуючий код (принцип відкритості закритості).

5.3 Хід роботи

Діаграма класів:

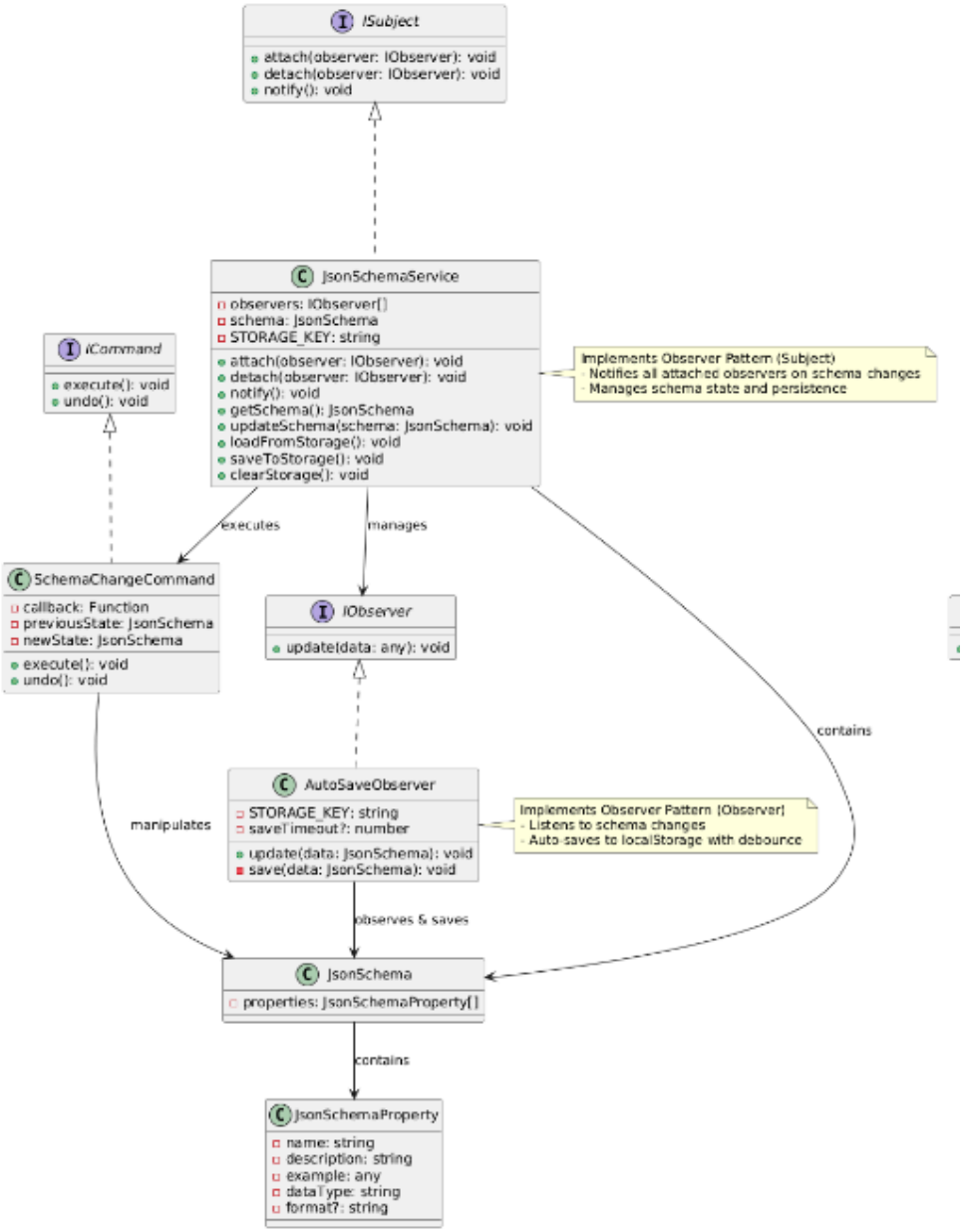


Рис 1. Діаграма класів системи

Фрагменти коду:

```
import { Injectable } from '@angular/core';
import { ICommand } from
'../../core/models/command.model';

@Injectable({
 providedIn: 'root'
})
export class HistoryService {
 private undoStack: ICommand[] = [];
 private redoStack: ICommand[] = [];

 executeCommand(command: ICommand): void {
   command.execute();
   this.undoStack.push(command);
   this.redoStack = [];
 }

 undo(): void {
   const command = this.undoStack.pop();
   if (command) {
     command.undo();
     this.redoStack.push(command);
   }
 }

 redo(): void {
   const command = this.redoStack.pop();
   if (command) {
```

```
      command.execute();
      this.undoStack.push(command);
    }
  }

  canUndo(): boolean {
    return this.undoStack.length > 0;
  }

  canRedo(): boolean {
    return this.redoStack.length > 0;
  }

  clear(): void {
    this.undoStack = [];
    this.redoStack = [];
  }
}

import { ICommand } from
'../../core/models/command.model';
import { JsonSchema } from
'../../core/models/json-schema.model';

export class SchemaChangeCommand implements ICommand
{
  constructor(
    private callback: (schema: JsonSchema) => void,
    private previousState: JsonSchema,
```

```
    private newState: JsonSchema
  ) {}

  execute(): void {
    this.callback(this.newState);
  }

  undo(): void {
    this.callback(this.previousState);
  }
}

export interface ICommand {
  execute(): void;
  undo(): void;
}
```

У реалізації використано патерн Command, який інкапсулює зміну стану JSON-схеми в окремий клас SchemaChangeCommand. Кожна команда містить як новий стан, так і попередній, а також callback, який застосовує цей стан. Сервіс HistoryService не знає деталей змін — він просто викликає execute() при виконанні команди та undo() при відміні. Команди зберігаються у двох стеках (undo/redo), що дозволяє керувати історією змін незалежно від логіки самої операції.

5.4 Висновок

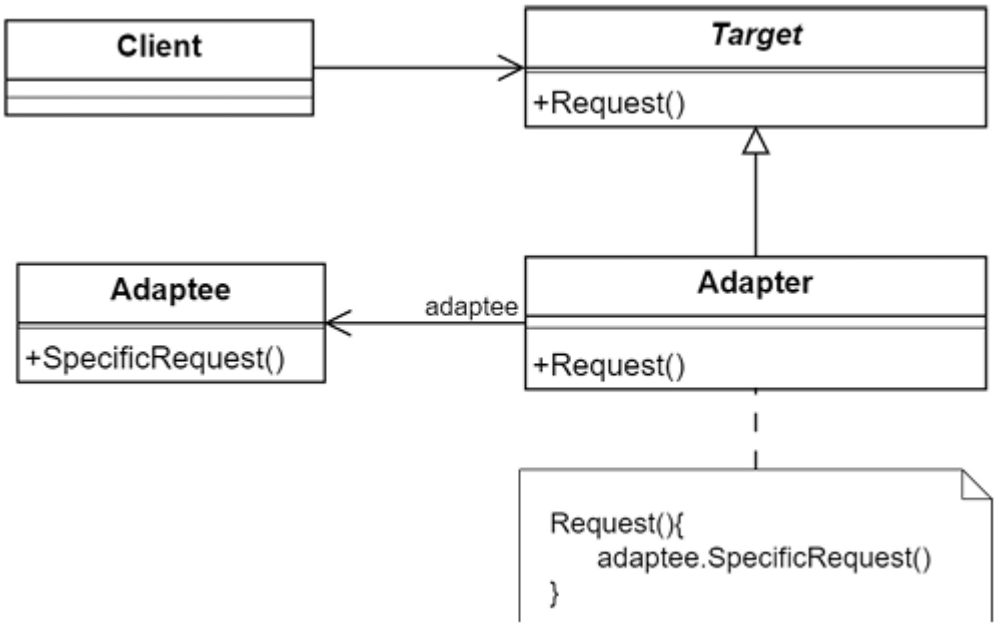
Під час даної роботи було реалізовано патерн Команда у додатку JSON Tool.

5.5 Контрольні запитання

1. **Яке призначення шаблону «Адаптер»?**

   Шаблон «Адаптер» дозволяє узгодити несумісні інтерфейси, щоб один клас міг використовувати інший, не змінюючи їх внутрішньої реалізації.

2. **Нарисуйте структуру шаблону «Адаптер».**



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

   До шаблону входять: Клієнт, Цільовий інтерфейс, Адаптер та Адаптований клас. Клієнт викликає методи цільового інтерфейсу, а адаптер переводить ці виклики у формат, який розуміє адаптований клас.

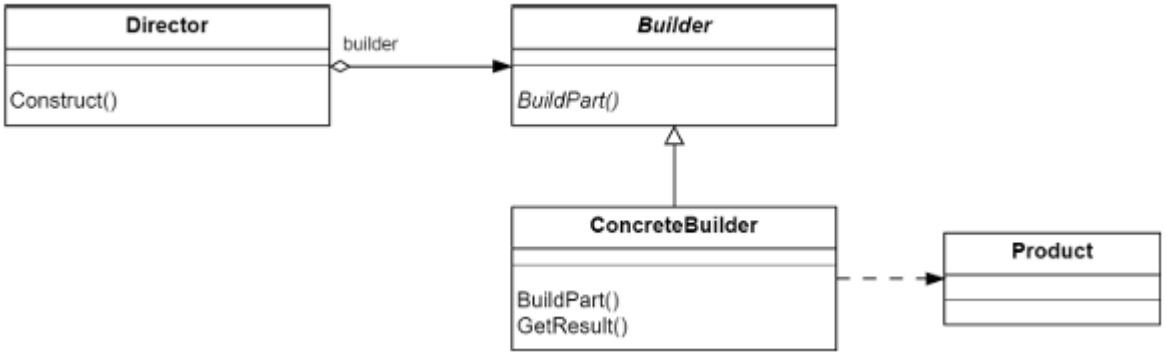
4. **Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?**

   Адаптер на рівні об’єктів використовує композицію і працює через посилання на адаптований об’єкт, тоді як адаптер на рівні класів використовує спадкування і підмінює інтерфейси через наслідування.

5. **Яке призначення шаблону «Будівельник»?**

Шаблон «Будівельник» застосовується для поетапного створення складних об’єктів, відокремлюючи процес побудови від їх представлення.

6. **Нарисуйте структуру шаблону «Будівельник».**



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

До шаблону входять: Директор, Інтерфейс Будівельника, Конкретний Будівельник, Продукт. Директор керує процесом побудови, будівельник визначає кроки створення, а результатом є готовий продукт.

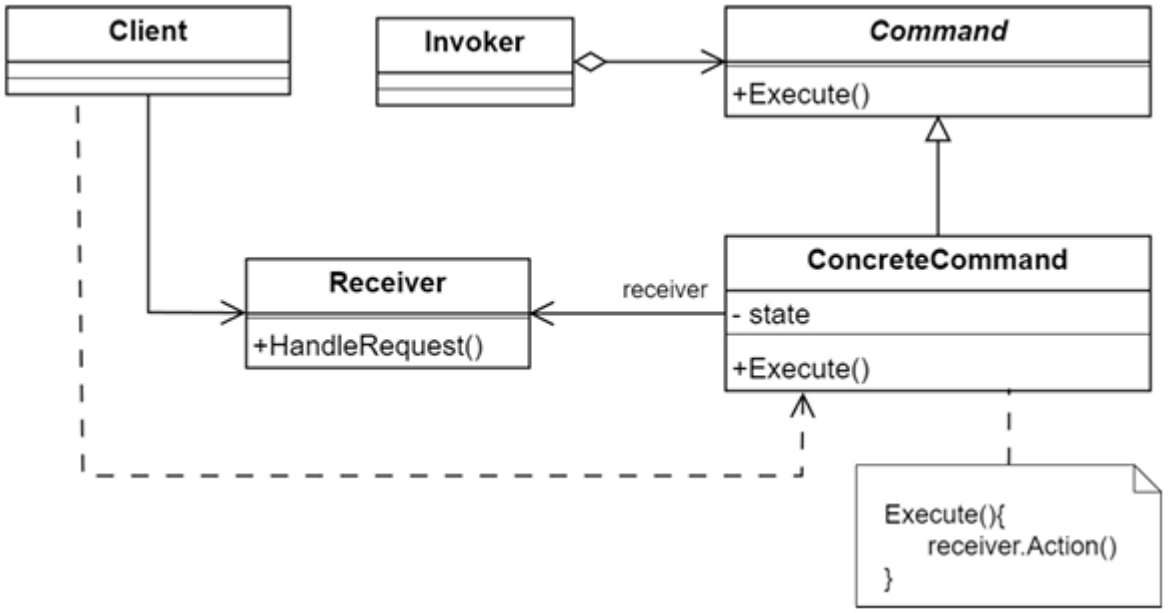
8. **У яких випадках варто застосовувати шаблон «Будівельник»?**

Будівельник варто застосовувати, коли об’єкт має багато варіантів конфігурації, складну структуру або коли потрібно контролювати порядок створення частин.

9. **Яке призначення шаблону «Команда»?**

Шаблон «Команда» інкапсулює дію у вигляді об’єкта, що дозволяє виконувати, відміняти та зберігати операції.

10. **Нарисуйте структуру шаблону «Команда».**



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

Основні класи: Інтерфейс Команди, Конкретні Команди, Одержувач, Інвокер і Клієнт. Клієнт створює команду, інвокер її викликає, а одержувач виконує необхідну дію.

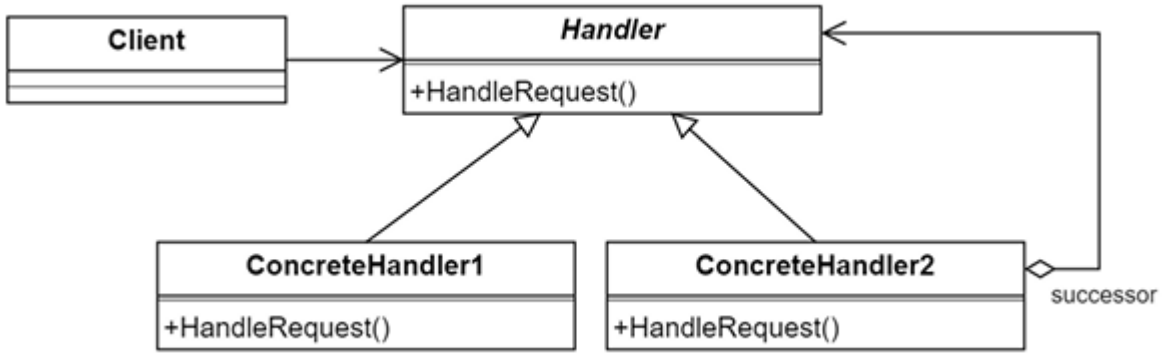
12. **Розкажіть як працює шаблон «Команда».**

Кожна дія оформлюється у вигляді об’єкта-команди з методами execute() та undo(). Інвокер викликає ці методи, не знаючи деталей реалізації, а самі команди зберігаються в історії для підтримки undo/redo.

13. **Яке призначення шаблону «Прототип»?**

Прототип дозволяє створювати нові об’єкти шляхом копіювання існуючих, не використовуючи конструктор.

14. **Нарисуйте структуру шаблону «Прототип».**



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

До шаблону входять: Інтерфейс Прототипа і Конкретні Прототипи. Клієнт викликає метод clone(), який повертає новий об’єкт, скопійований з існуючого.

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

Прикладами є обробка HTTP middleware, фільтрація подій у UI, логування, послідовні перевірки прав доступу, маршрутизація повідомлень та обробка винятків у складних системах.