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 «Теорія розробки прогрманого забезпечення»

Тема: «Патерни проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Патерни проектування.

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

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Шаблон «Command» Призначення патерну: Шаблон "command" (команда) перетворить звичайний виклик методу в клас [6]. 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 аттрибут 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).

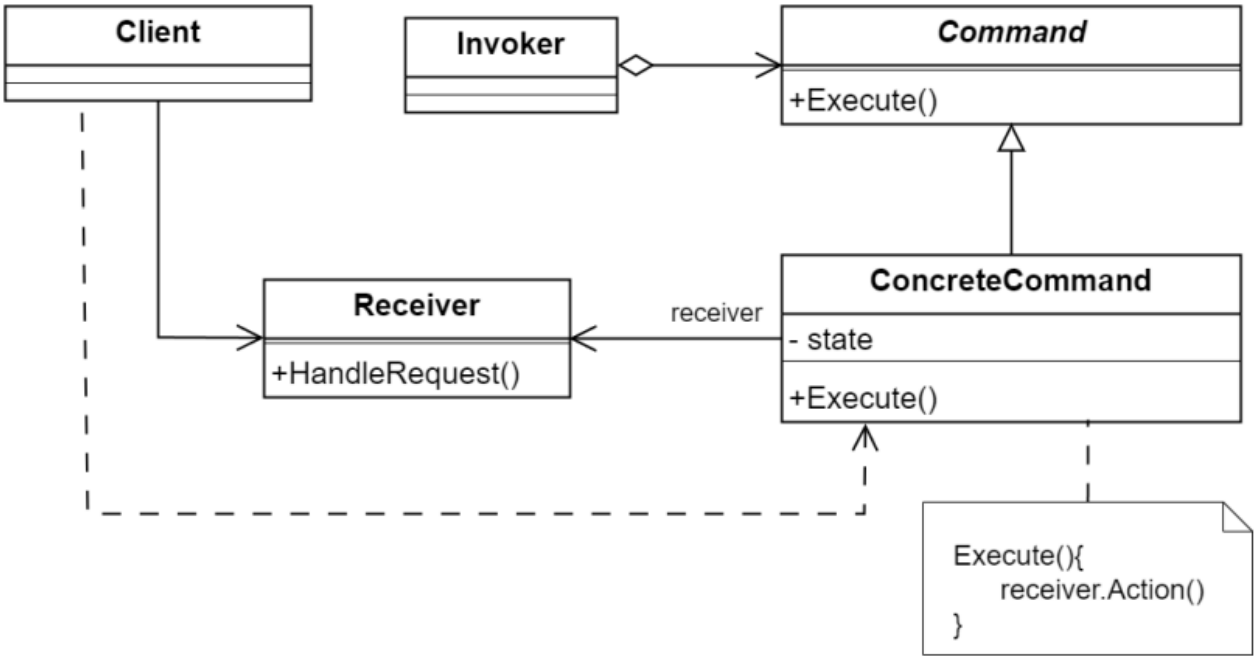


Рисунок 5.3. Структура патерну Команда

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 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.

Додатковим викликом буде реалізувати автоматизоване тестування такої системи, тому що нам необхідно емулювати натискання кнопок користувачем.

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.

При такій реалізації також набагато простіше організувати тестування застосунку, тому що ми тепер не потрібно емулювати дії користувача, а достатньо протестувати шар логіки обробки використовуючи модульні тести (unit tests).

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 відкритостізакритості).

Хід роботи

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.

Реалізація шаблону проєктування для майбутньої системи

Ми використовуємо патерн Command для інкапсуляції запитів користувача на виконання скриптів PowerShell у вигляді окремих об'єктів. Замість того, щоб виконувати логіку запуску процесів та обробки результатів безпосередньо в контролері, ми перетворюємо кожен запит на об'єкт-команду (PowerShellExecuteCommand), що містить усю необхідну інформацію для дії. Це дозволяє відділити ініціатора запиту (веб-інтерфейс) від механізму його виконання, а також централізовано керувати історією через клас-інвокер (CommandInvoker), який автоматично зберігає ввід та вивід команд для відображення у браузері, не ускладнюючи при цьому код контролера.

```
package org.kpi.pattern.command;

public interface Command {  3 usages  1 implementation  new *
    String execute();  1 usage  1 implementation  new *
}
```

Рис. 1 – Код інтерфейсу Command

```java
/**
 * Receiver (Одержувач).
 * Знає, як фактично запустити процес PowerShell.
 */
@Service  6 usages  new *
public class PowerShellService {

    public String runCommand(String commandText) {  1 usage  new *
        StringBuilder output = new StringBuilder();
        try {
            ProcessBuilder builder = new ProcessBuilder( ...command: "powershell.exe", "/c", commandText);
            builder.redirectErrorStream(true);
            Process process = builder.start();

            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(process.getInputStream(), Charset.forName( charsetName: "CP866"))
            );

            String line;
            while ((line = reader.readLine()) != null) {
                output.append(line).append("\n");
            }

            process.waitFor();
        } catch (Exception e) {
            return "Error executing command: " + e.getMessage();
        }
        return output.toString();
    }
}
```

Рис. 2 – Код класу PowerShellService

```java
public class PowerShellExecuteCommand implements Command {  5 usages  new *

    private final PowerShellService service;  2 usages
    private final String script;  3 usages

    public PowerShellExecuteCommand(PowerShellService service, String script) {  1 usage  new *
        this.service = service;
        this.script = script;
    }

    @Override  1 usage  new *
    public String execute() {
        // Делегуємо виконання сервісу
        return service.runCommand(script);
    }

    public String getCommandText() {  1 usage  new *
        return script;
    }
}
```

Рис. 3 – Код класу PowerShellExecuteCommand

```java
/**
 * Invoker (Ініціатор).
 * Викликає виконання команди та зберігає історію.
 */
@Service  3 usages  new *
public class CommandInvoker {

    @Getter
    private final List<String> history = new ArrayList<>();

    public void executeCommand(Command command) {  1 usage  new *
        if (command instanceof PowerShellExecuteCommand psCmd) {
            history.add("PS User> " + psCmd.getCommandText());
        }

        String result = command.execute();

        if (result != null && !result.isEmpty()) {
            history.add(result);
        } else {
            history.add("");
        }
    }

    // Метод для очищення історії (опціонально)
    public void clearHistory() {  no usages  new *
        history.clear();
    }
}
```

Рис. 4 – Код класу CommandInvoker

Зображення структури шаблону

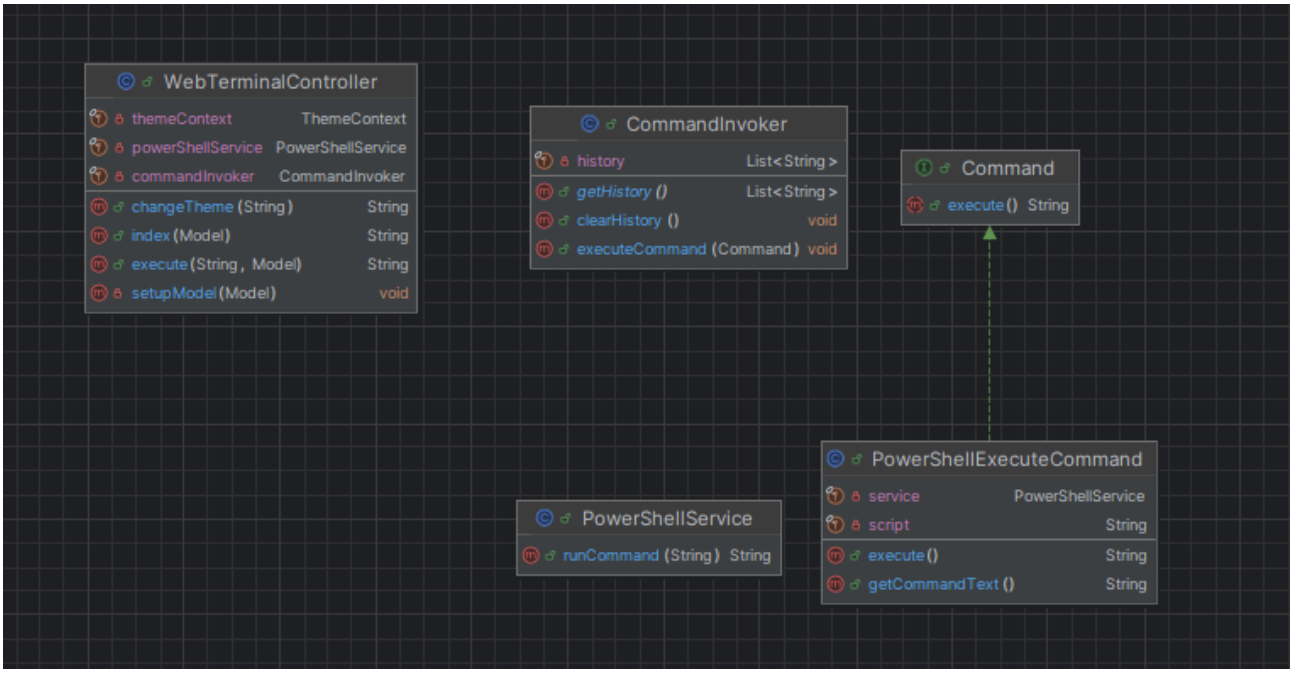


Рис. 5 – Структура шаблону Command

Структура реалізації:

- Command – інтерфейс, що визначає єдиний метод execute() для виконання команди, слугуючи загальним контрактом для всіх типів операцій у терміналі.
- PowerShellExecuteCommand – конкретна реалізація команди, яка інкапсулює запит користувача (текст скрипта) та пов’язує його з виконавцем (PowerShellService) для безпосереднього запуску.
- PowerShellService (Receiver) – клас-виконавець (одержувач), що містить низькорівневу бізнес-логіку запуску процесу powershell.exe та зчитування результату виконання.
- CommandInvoker (Invoker) – ініціатор (інвокер), який не знає деталей виконання, але ініціює запуск команди та відповідає за ведення журналу (історії) введених команд і отриманих результатів.
- WebTerminalController (Client) – клієнт, який створює об’єкт команди, налаштовує його (передаючи необхідні дані та посилання на одержувача) і передає інвокеру для виконання.

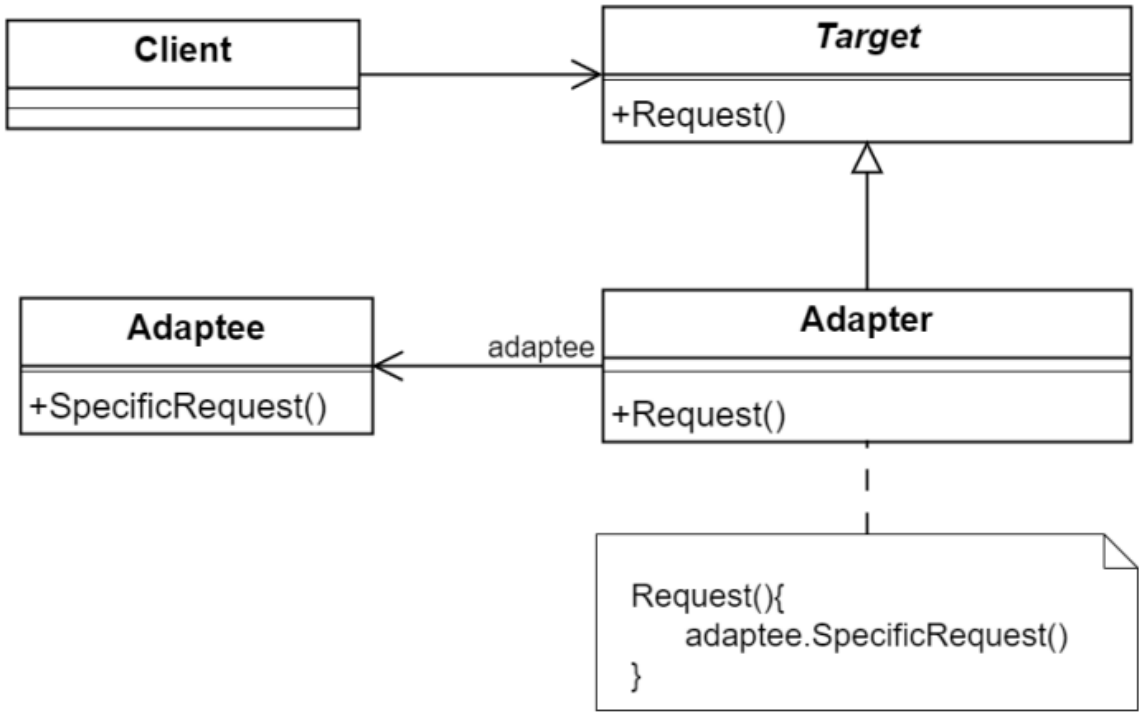
Такий підхід дозволяє інкапсулювати запити до системи у вигляді окремих об'єктів, відокремлюючи об'єкт, що ініціює запит (веб-контролер), від об'єкта, що знає, як його виконати (сервіс). Це дає змогу централізовано керувати історією операцій, легко додавати нові типи команд та змінювати механізм їх обробки без впливу на клієнтський код.

Питання до лабораторної роботи

1. Яке призначення шаблону «Адаптер»?

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.

2. Нарисуйте структуру шаблону «Адаптер».



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

Target (Ціль): Визначає інтерфейс, який використовує клієнт.

Client (Клієнт): Взаємодіє з об'єктами, що дотримуються інтерфейсу Target.

Adaptee (Адаптований): Визначає існуючий інтерфейс, який потрібно адаптувати (наприклад, сторонній клас або застарілий код).

Adapter (Адаптер): Реалізує інтерфейс Target і зберігає посилання на екземпляр Adaptee. Він виступає посередником, адаптуючи виклики клієнта до інтерфейсу адаптованого класу.

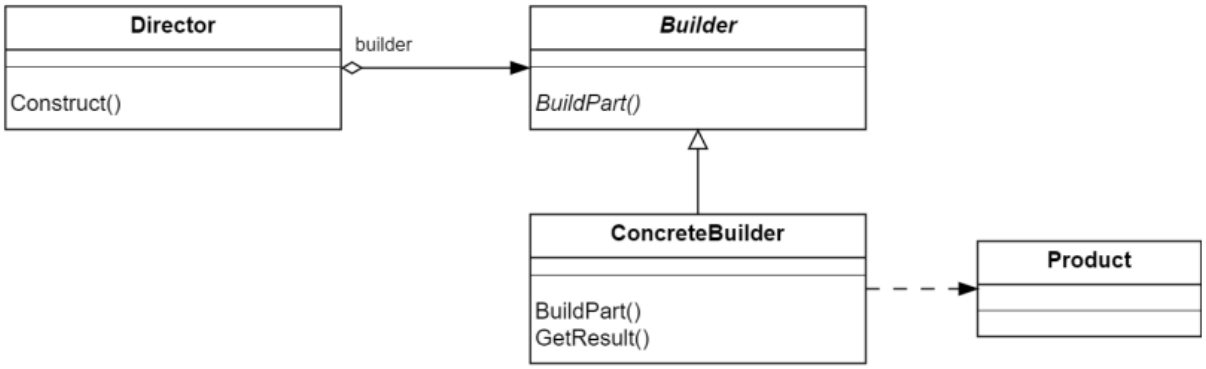
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

Різниця полягає у способі інтеграції адаптованого функціоналу: адаптер на рівні класів використовує успадкування (реалізує інтерфейс цілі та успадковує клас адаптованого об'єкта), що дозволяє перевизначати його методи, але прив'язує до конкретної реалізації. Водночас адаптер на рівні об'єктів базується на композиції (зберігає посилання на екземпляр адаптованого об'єкта), що є більш гнучким підходом, оскільки дозволяє адаптувати як сам клас, так і його підкласи

5. Яке призначення шаблону «Будівельник»?

Шаблон «Builder» (Будівельник) використовується для відділення процесу створення об'єкту від його представлення.

6. Нарисуйте структуру шаблону «Будівельник».



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

Director (Розпорядник): Клас, який керує процесом створення об'єкта. Він викликає методи будівельника в певному порядку для побудови продукту.

Builder (Будівельник): Абстрактний інтерфейс, що оголошує методи для створення частин об'єкта-продукту (наприклад, BuildPart()).

ConcreteBuilder (Конкретний будівельник): Клас, який реалізує інтерфейс Builder. Він конструює та збирає конкретні частини продукту, а також надає метод для отримання готового результату (наприклад, GetResult()).

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

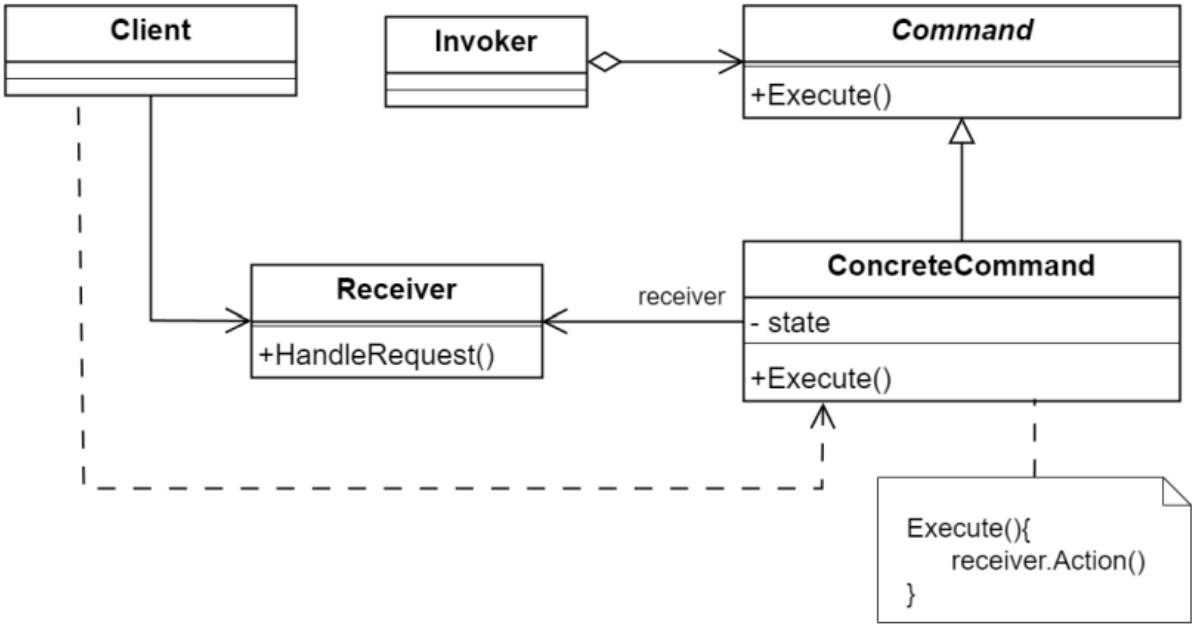
8. У яких випадках варто застосовувати шаблон «Будівельник»"?

Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Webсторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати декілька різних форм створення (наприклад, при конвертації тексту з формату у формат).

9. Яке призначення шаблону «Команда»?

Шаблон "command" (команда) перетворить звичайний виклик методу в клас. Таким чином дії в системі стають повноправними об'єктами.

10.Нарисуйте структуру шаблону «Команда».



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

Command (Команда): Інтерфейс, що оголошує метод для виконання операції (зазвичай Execute()).

ConcreteCommand (Конкретна команда): Клас, що реалізує інтерфейс Command. Він визначає зв'язок між дією та одержувачем (Receiver), викликаючи відповідні методи одержувача для виконання запиту.

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

Invoker (Ініціатор): Зберігає команду та ініціює запит, викликаючи метод Execute() у команди. Він не знає нічого про конкретну реалізацію команди.

Receiver (Одержувач): Об'єкт, який знає, як виконувати операції, пов'язані з виконанням запиту.

12.Розкажіть як працює шаблон «Команда».

Він перетворює звичайний виклик методу в окремий об'єкт-клас. Це дозволяє відділити об'єкт, що ініціює дію (наприклад, кнопка в меню), від об'єкта, що знає, як цю дію виконати (бізнес-логіка).

Механізм роботи:

Сама по собі команда не виконує фактичної роботи. Вона зберігає посилання на одержувача (Receiver) та перенаправляє йому запит на виконання дії.

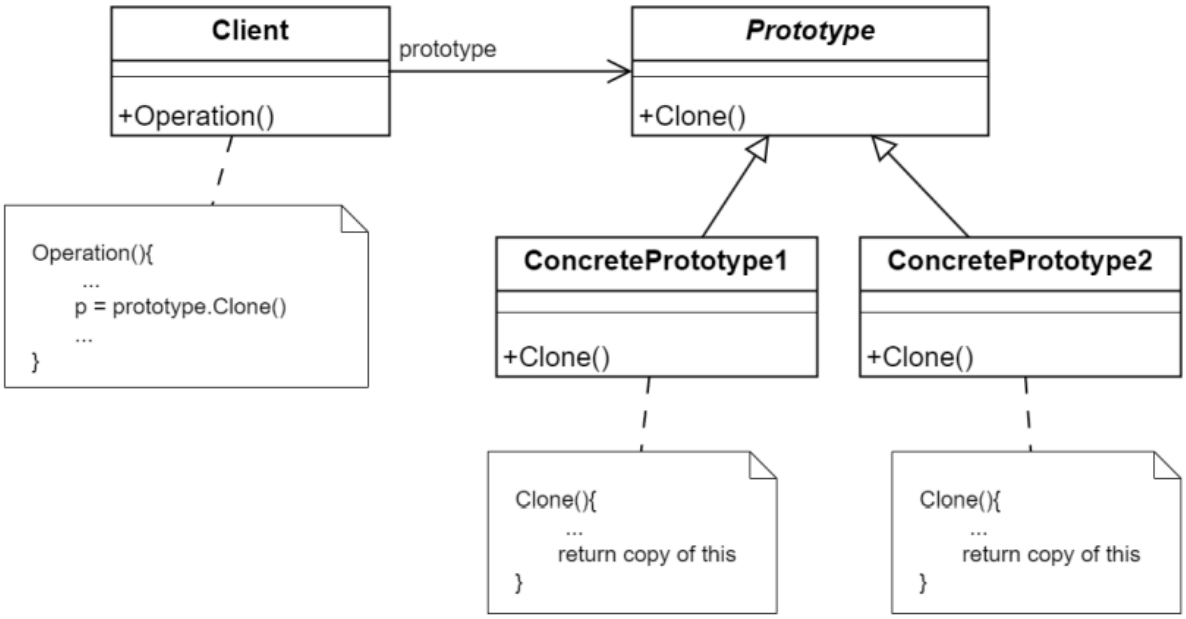
Коли Invoker (ініціатор) викликає метод виконання у команди (наприклад, Execute()), команда звертається до Receiver і запускає потрібну операцію (receiver.Action()) .

Додаткові можливості: Оскільки команда є об'єктом, вона може зберігати свій стан та параметри. Це дозволяє реалізувати такі функції, як скасування дій (Undo), логування історії команд, створення черги запитів або відкладений запуск .

13. Яке призначення шаблону «Прототип»?

Шаблон «Prototype» (Прототип) використовується для створення об'єктів за «шаблоном» (чи «кресленням», «ескізом») шляхом копіювання шаблонного об'єкту, який називається прототипом.

14. Нарисуйте структуру шаблону «Прототип».



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

Prototype (Прототип): Інтерфейс або абстрактний клас, який оголошує метод Clone() для клонування самого себе.

ConcretePrototype (Конкретний прототип): Клас, який реалізує інтерфейс Prototype і визначає операцію клонування. Він створює точну копію свого екземпляра. На схемі це ConcretePrototype1 та ConcretePrototype2.

Client (Клієнт): Створює нові об'єкти, звертаючись до прототипу із запитом клонувати себе.

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

Приклад із реального життя: Процес підписання документа в компанії. Документ проходить шлях від співробітника, який його склав, через менеджера та начальника відділу аж до головного керівника, який ставить фінальний підпис.

Приклад у розробці програмного забезпечення (UI): Формування контекстного меню для складних форм із вкладеними компонентами.

Висновок

Під час виконання лабораторної роботи я вивчив структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчився застосовувати їх в реалізації програмної системи. Реалізував шаблон «Command» у проєкті Power Shell термінал.