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 «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

«2. HTTP-сервер»

Виконав:
студент групи - ІА-32
Непотачев Іван Дмитрович

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проектування.

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

Тема проєкту: HTTP-сервер (state, builder, factory method, mediator, composite, p2p) Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Зміст

Теоретичні відомості 2
Хід роботи 3
Реалізація шаблону проєктування для майбутньої системи 3
Зображення структури шаблону 7
Висновки 9
Питання до лабораторної роботи 9

Теоретичні відомості

Шаблон «Builder»
Призначення патерну: Шаблон «Builder» (Будівельник) використовується для відділення процесу створення об'єкту від його представлення [6]. Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Web-сторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати декілька різних форм створення (наприклад, при конвертації тексту з формату у формат).
Проблема: Візьмемо процес побудови відповіді на запит web-сервера. Побудова складається з наступних частин: додавання стандартних заголовків (дата/час, ім'я сервера, інш.), код статусу (після пошуку відповідної сторінки на сервері), заголовки відповіді (тип вмісту, інш.), утримуване, інше.

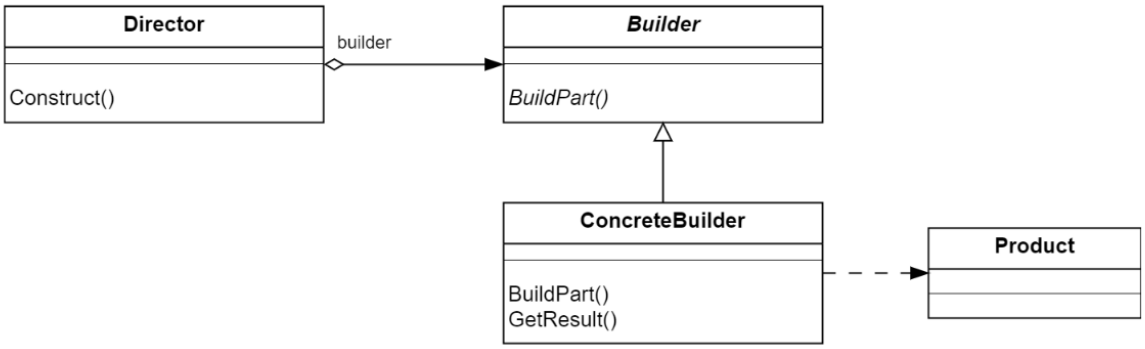


Рисунок 5.2. Структура патерну Builder

Рішення: Кожен з цих етапів може бути абстрагований в окремий метод будівельника. Це дасть наступні вигоди:
- Гнучкіший контроль над процесом створення сторінки;
- Незалежність від внутрішніх змін – наприклад, зміна назви сервера не сильно порушить процес побудови відповіді;
Переваги та недоліки:
+ Дозволяє використовувати один і той самий код для створення різноманітних продуктів.
- Клієнт буде прив'язаний до конкретних класів будівельників, тому що в інтерфейсі будівельника може не бути методу отримання результату.

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

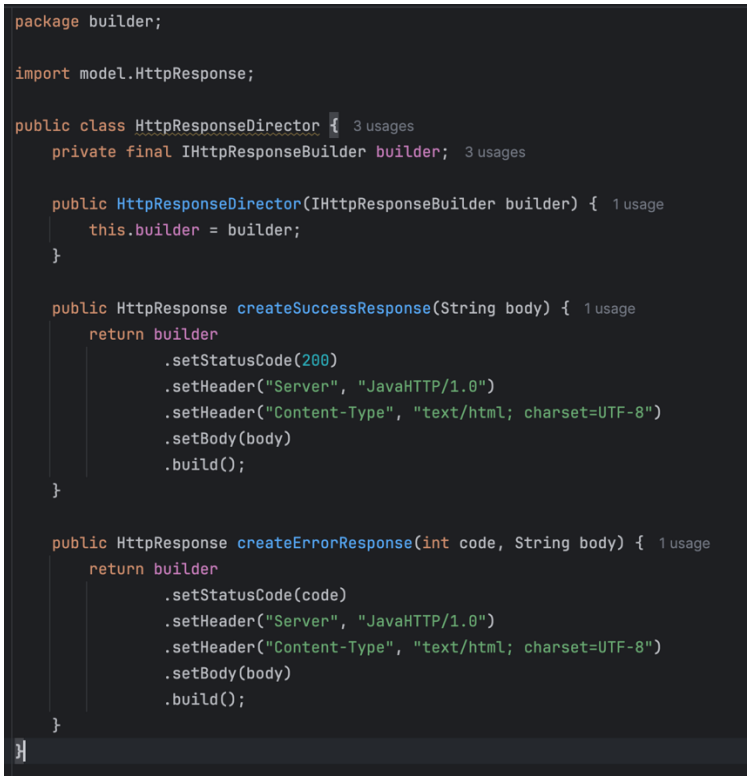
Для реалізації HTTP-сервера використано шаблон проєктування Builder, оскільки він забезпечує зручне, поетапне та безпечне створення складних об'єктів запитів і відповідей (HttpRequest та HttpResponse), які містять велику кількість параметрів. Це дозволяє уникнути перевантажених конструкторів і підвищує гнучкість при формуванні HTTP-повідомлень.

Шаблон Builder реалізовано у класах HttpRequestBuilder та HttpResponseBuilder, які відповідають за поступове налаштування всіх

необхідних параметрів перед створенням кінцевого об’єкта. Такий підхід є особливо ефективним у контексті роботи HTTP-сервера, де кожен запит і відповідь можуть мати різну структуру заголовків, тіла та статусів.

```java
package builder;

import model.HttpResponse;
import java.util.HashMap;
import java.util.Map;

public class HttpResponseBuilder implements IHttpResponseBuilder {  2 usages
    private int statusCode;  2 usages
    private String statusMessage;  5 usages
    private Map<String, String> headers = new HashMap<>();  2 usages
    private String body = "";  2 usages


    @Override  2 usages
    public IHttpResponseBuilder setStatusCode(int code) {
        this.statusCode = code;
        switch (code) {
            case 200 -> this.statusMessage = "OK";
            case 404 -> this.statusMessage = "Not Found";
            case 503 -> this.statusMessage = "Service Unavailable";
            default -> this.statusMessage = "Unknown";
        }
        return this;
    }

    @Override  4 usages
    public IHttpResponseBuilder setHeader(String key, String value) {
        headers.put(key, value);
        return this;
    }

    @Override  2 usages
    public IHttpResponseBuilder setBody(String body) {
        this.body = body;
        return this;
    }

    @Override  2 usages
    public HttpResponse build() {
        return new HttpResponse(statusCode, statusMessage, headers, body);
    }
```

Рис. 1 – Код класу HTTP-Response Builder



```java
package builder;

import model.HttpResponse;

public class HttpResponseDirector {  3 usages
    private final IHttpResponseBuilder builder;  3 usages

    public HttpResponseDirector(IHttpResponseBuilder builder) {  1 usage
        this.builder = builder;
    }

    public HttpResponse createSuccessResponse(String body) {  1 usage
        return builder
                .setStatusCode(200)
                .setHeader("Server", "JavaHTTP/1.0")
                .setHeader("Content-Type", "text/html; charset=UTF-8")
                .setBody(body)
                .build();
    }

    public HttpResponse createErrorResponse(int code, String body) {  1 usage
        return builder
                .setStatusCode(code)
                .setHeader("Server", "JavaHTTP/1.0")
                .setHeader("Content-Type", "text/html; charset=UTF-8")
                .setBody(body)
                .build();
    }
}
```

Рис. 2 – Код класу HTTP-Response Director

Застосування цього шаблону забезпечує:

1. Поетапне створення складних об’єктів: Класи HttpRequestBuilder і HttpResponseBuilder дозволяють поступово задавати параметри, такі як метод запиту, URL, статус-код, заголовки, тіло повідомлення тощо. Це робить створення HTTP-повідомлень інтуїтивно зрозумілим і контрольованим процесом.
2. Зменшення кількості перевантажених конструкторів: Без Builder’а довелося б створювати конструктори з різною кількістю параметрів (наприклад, лише статус, або статус + заголовки + тіло). Використання Builder-підходу дозволяє уникнути цієї проблеми й робить код чистішим.
3. Підвищення гнучкості та розширюваності: Якщо в майбутньому потрібно буде додати нові параметри HTTP-запиту чи відповіді, достатньо додати нові методи у Builder без зміни існуючих класів HttpRequest або HttpResponse.
4. Зручність використання та читабельність коду: Завдяки підтримці «ланцюжкового» виклику методів, розробник може легко створити об’єкт у зрозумілому вигляді:

У нашому випадку шаблон Builder забезпечує чітке розділення процесу побудови HTTP-повідомлень від їхньої внутрішньої реалізації. Це

дозволяє зосередитись на логіці обробки запитів, не турбуючись про технічні деталі формування структури повідомлення.

Зображення структури шаблону

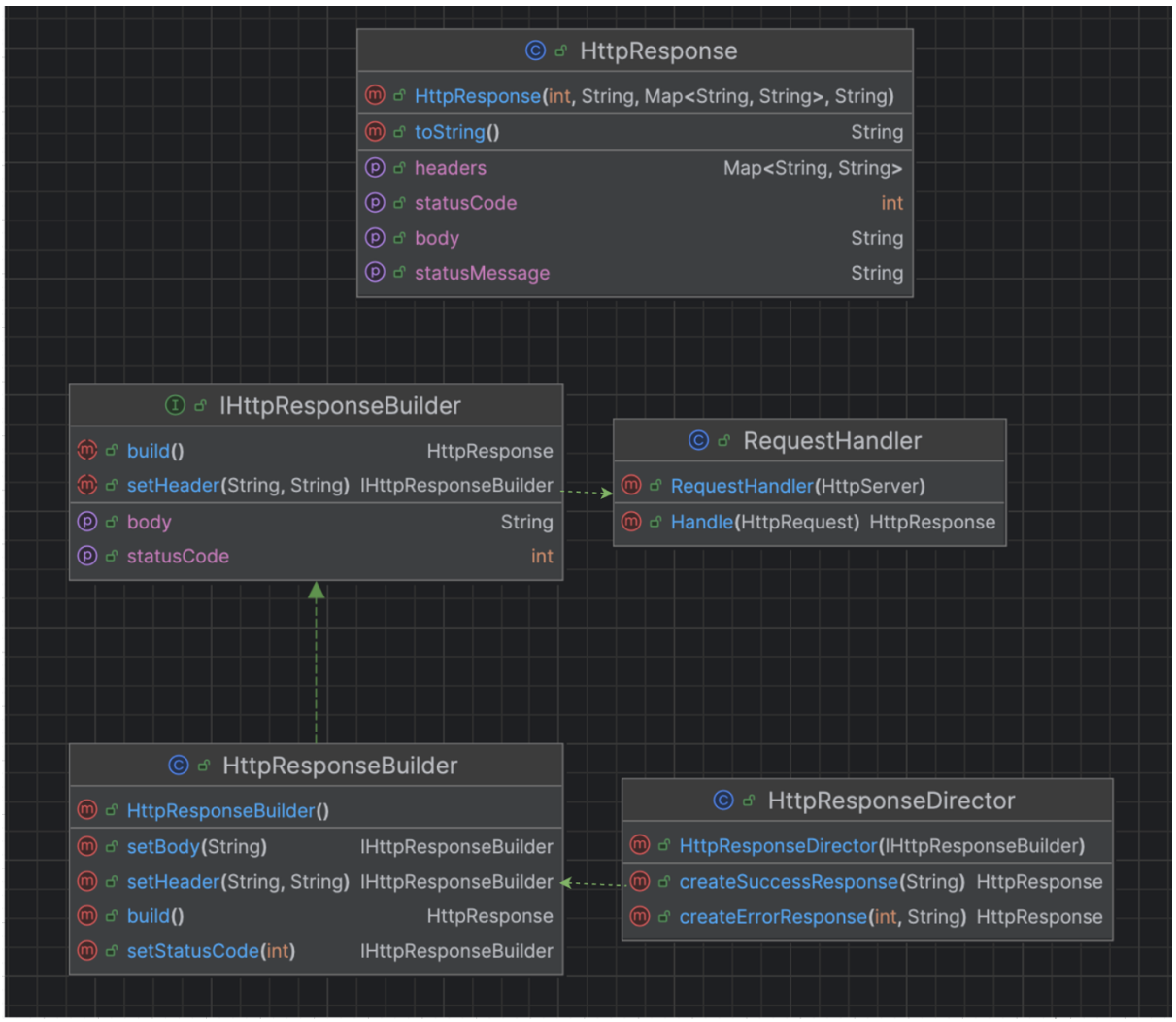


Рис. 2 – Структура шаблону Builder

Структура реалізації:

• HttpRequest — клас, що представляє HTTP-запит і містить основні поля: метод, URL, заголовки та тіло запиту.

• HttpRequestBuilder — клас-будівник для поетапного створення об’єкта HttpRequest. Дає змогу встановлювати метод, шлях, параметри, заголовки тощо перед побудовою кінцевого об’єкта.

• HttpResponse — клас, що описує HTTP-відповідь і зберігає код стану, тіло та заголовки.

- HttpResponseBuilder — клас-будівник, який дозволяє задавати статус-код, повідомлення, тіло та заголовки у зручній формі.

- HttpServer — використовує ці класи для побудови запитів і відповідей у процесі обробки HTTP-трафіку.

Такий підхід забезпечує гнучкість і зрозумілу структуру коду, дозволяє створювати запити та відповіді з різними параметрами, уникаючи складних і перевантажених конструкторів.

Однак у випадках, коли об’єкти є простими та містять лише кілька параметрів, використання Builder може бути надмірним, адже процес побудови об’єкта може бути реалізований простим викликом конструктора. Попри це, для систем, що працюють із великою кількістю змінних параметрів (як HTTP-запити/відповіді), Builder залишається оптимальним рішенням, оскільки спрощує розширення класів, забезпечує чистоту та зрозумілість коду та дозволяє створювати об’єкти з різними конфігураціями без дублювання логіки.

Посилання на репозиторій: https://github.com/IvanGodPro24/trpz
Посилання на звіти: https://github.com/IvanGodPro24/trpz_reports

Висновки

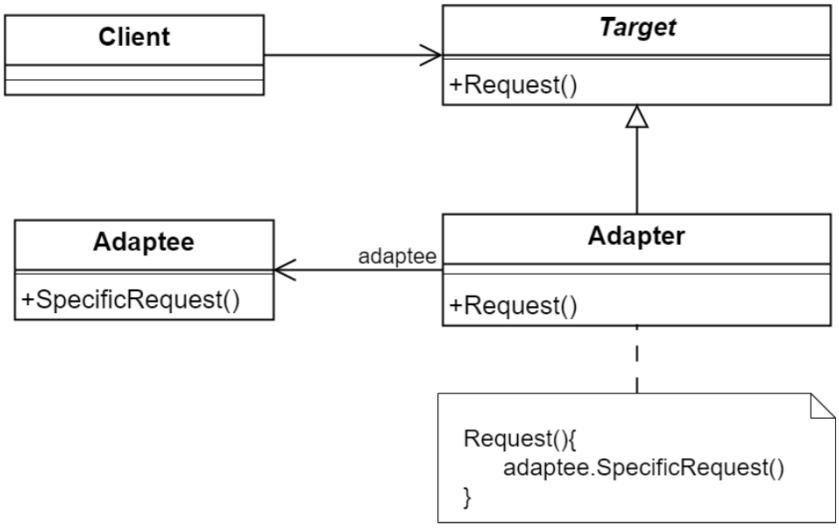
Висновки: під час виконання лабораторної роботи, ми вивчили структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчилися застосовувати їх в реалізації програмної системи.

Питання до лабораторної роботи

1. Яке призначення шаблону «Адаптер»?

Шаблон «Адаптер» (Adapter) дозволяє об’єктам з несумісними інтерфейсами працювати разом. Він перетворює інтерфейс одного класу у вигляд, сумісний з інтерфейсом іншого класу.

2. Нарисуйте структуру шаблону «Адаптер».



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

Target — інтерфейс, який очікує клієнт.

Adaptee — існуючий клас із несумісним інтерфейсом.

Adapter — перетворює виклики від клієнта у виклики методів Adaptee.

Client — працює лише з Target.

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

На рівні об’єктів адаптер містить об’єкт Adaptee (через композицію).

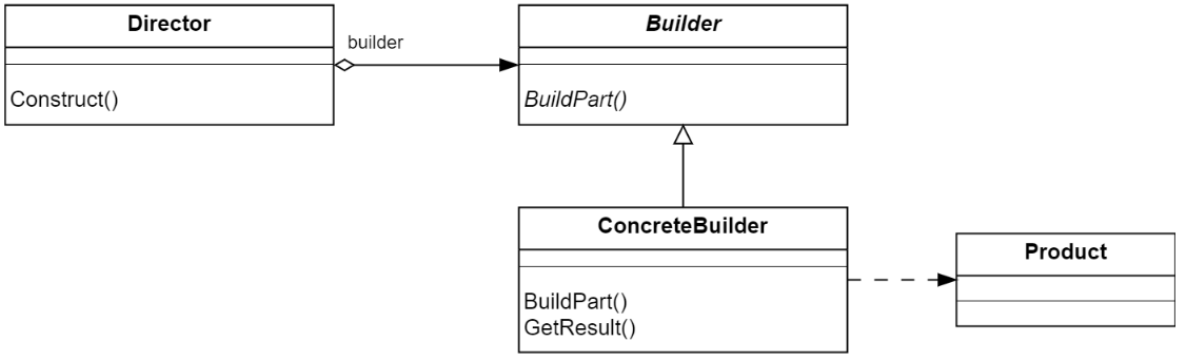
На рівні класів адаптер наслідує як Target, так і Adaptee (через множинне успадкування).

Перший варіант гнучкіший, другий — жорсткіше пов’язаний із класами.

5. Яке призначення шаблону «Будівельник»?

Шаблон «Будівельник» (Builder) відокремлює процес створення складного об’єкта від його представлення, щоб той самий процес створення міг давати різні об’єкти.

6. Нарисуйте структуру шаблону «Будівельник».



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

Director — керує процесом побудови.

Builder — визначає інтерфейс для створення частин продукту.

ConcreteBuilder — реалізує конкретні етапи побудови.

Product — кінцевий об’єкт, який створюється.

Director викликає методи Builder, який поступово створює Product.

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

Коли об’єкт має складну структуру або багато кроків створення.

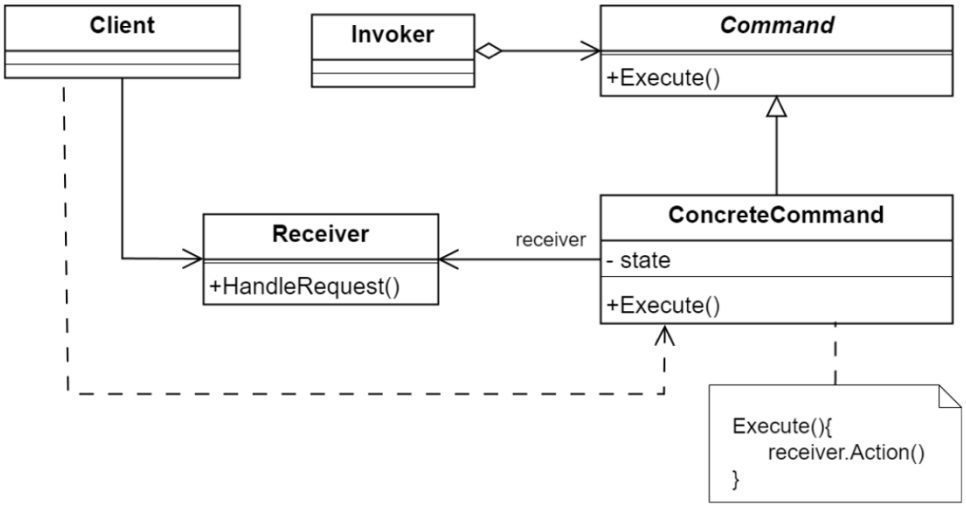
Коли потрібно створювати різні представлення одного об’єкта.

Коли потрібно ізолювати код створення від коду представлення.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» (Command) інкапсулює запит у вигляді об’єкта, дозволяючи зберігати, передавати, скасовувати або відкладати виконання операцій.

10. Нарисуйте структуру шаблону «Команда».



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

Command — інтерфейс команди.

ConcreteCommand — реалізує конкретну операцію.

Invoker — ініціює виконання команди.

Receiver — виконує фактичну дію.

Client — створює команди та задає Invoker-у.

Client створює ConcreteCommand, Invoker викликає її метод execute(), який звертається до Receiver.

12. Розкажіть як працює шаблон «Команда».

1. Клієнт створює об’єкт команди, пов’язаний із певним виконавцем (Receiver).

2. Команда передається об’єкту Invoker.

3. Invoker викликає execute(), не знаючи деталей реалізації.

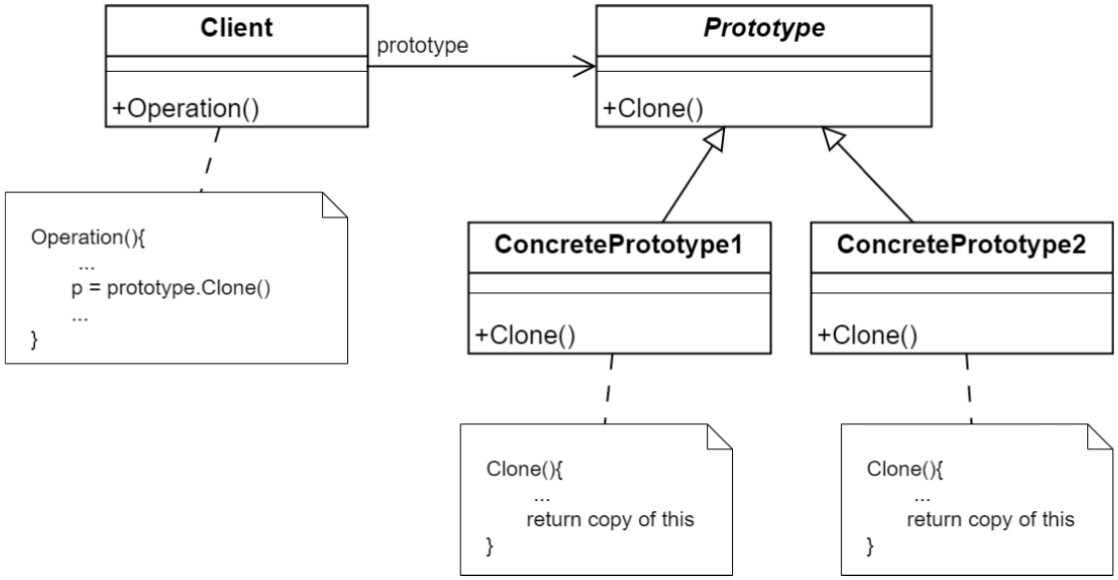
4. Команда виконує дію, викликаючи методи Receiver.

5. Команду можна скасувати або зберегти для повторного виконання.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» (Prototype) дозволяє створювати нові об’єкти, копіюючи існуючі прототипи, замість створення їх із нуля.

14. Нарисуйте структуру шаблону «Прототип».



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

Prototype — інтерфейс із методом clone().

ConcretePrototype — реалізує копіювання самого себе.

Client — створює нові об’єкти через копіювання прототипів.

Client викликає clone() у прототипу, отримуючи новий екземпляр з тими самими властивостями.

16. Які можна привести приклади використання шаблону «Ланцюжок

відповідальності»?

- Обробка подій у графічному інтерфейсі (натискання кнопки, події миші).
- Система обробки запитів у вебсерверах (middleware).
- Система підтримки користувачів: запит передається від оператора до менеджера.
- Обробка запитів на доступ: різні рівні безпеки перевіряють запит послідовно.