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»

«Web crawler»

Виконала
студентка групи – ІА–31
Горлач Дар’я Дмитрівна

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: Web crawler (proxy, chain of responsibility, memento, template method, composite, p2p) Веб–сканер повинен вміти розпізнавати структуру сторінок сайту, переходити за посиланнями, збирати необхідну інформацію про зазначений термін, видаляти не семантичні одиниці (рекламу, об'єкти javascript і т.д.), зберігати знайдені дані у вигляді структурованого набору html файлів вести статистику відвіданих сайтів і метадані.

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.

Зміст

Теоретичні відомості

Шаблон «Chain of Responsibility»

Призначення патерну: Шаблон «Chain of responsibility» (Ланцюжок відповідальності) частково можна спостерігати в житті, коли підписання відповідного документу проходить від його складання у одного із співробітників компанії через менеджера і начальника до головного начальника, який ставить свій підпис [5].

Проблема: Ви розробляєте систему зі складними UI формами, які містять багато вкладених один в один візуальних компонентів, по кліку правою кнопкою миші вам потрібно сформувати контекстне меню. В залежності від того на якому компоненті був виконаний клік мишкою, вміст контекстного меню повинен відрізнятися, також в контекстне меню потрібно добавляти пункти, якщо компонент, в який входить поточний, теж має свої пункти. Один із очевидних підходів – ви робити метод, який викликається по кліку мишки. В методі ви робити перевірки і якщо в даний момент конкретний елемент, добавляєте в контекстне меню підходящі до нього пункти, потім перевіряєте інші елементи і перевіряєте чи вони не є такими, що містять в собі вибраний елемент. Якщо так, то добавляєте в контекстне меню ще пункти.

З часом алгоритм формування меню стає більш складним, тому що добавляються нові елементи. Також з часом в алгоритмі залишилися перевірки на компоненти, які з форми були вже видалені.

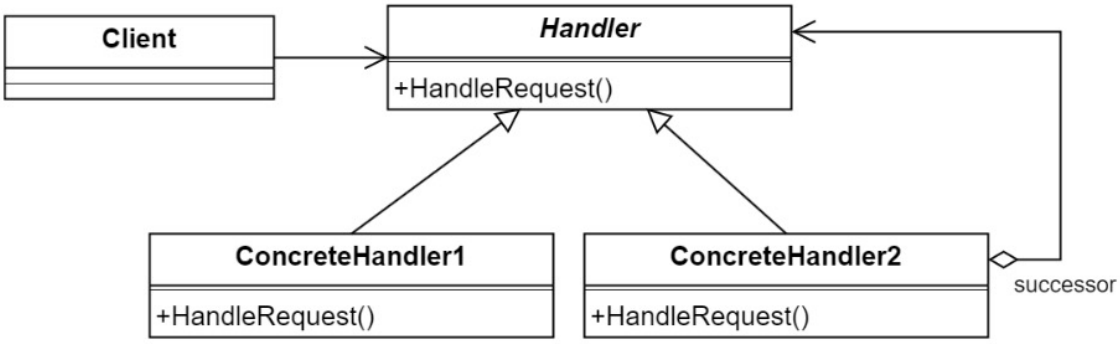


Рисунок 1. Структура патерну Ланцюжок відповідальності

Рішення: Для вирішення проблеми зі зростаючою складністю формування контестного меню підходить патерн «Ланцюжок відповідальності».

Основна ідея в тому, що нам не потрібно мати загальний метод формування контекстного меню. Ми можемо зробити загальний інтерфейс з методом UpdateContextMenu() і реалізувати цей метод в усіх візуальних компонентах. Додатково для візуальних компонентів добавляємо поле для переходу на "батьківський" елемент, якій в собі містить поточний візуальний компонент, а в реалізації UpdateContextMenu в кінці додаємо виклик цього метода у "батьківського" елемента. Якщо елемент не потребує додавання пунктів в контекстне меню, він просто викликає цей метод у наступного елемента в ланцюжку. Тепер обробка правого кліку мишки буде виглядати наступним чином: По кліку правою кнопкою миші викликаємо UpdateContextMenu на тексті, він, наприклад, додає пункт меню «Копіювати текст», далі викликає аналогічний метод у компонента «текст–блок» в якому цей текст знаходиться, але текст–блок нічого в контекстне меню не добавляє, а викликає UpdateContextMenu в елемента

Grid, в якому знаходиться текст–блок, а Grid при відпрацюванні метода добавляє пункт меню «Контестна допомога» і так далі.

Коли ми змінюємо структуру форми, то між компонентами змінюються зв'язки в ланцюжку, але самі алгоритми вже переробляти не потрібно. Якщо видаляється з форми якийсь компонент, то разом з ним видаляється і логіка пов'язана з цим компонентом. Таким чином складність роботи з контекстним меню залишається контрольованою.

Слід добавити що елемент, що обробляє виклик, не обов'язково повинен передавати виклик далі по ланцюжку. Це залежить від того, яку поведінку ви хочете отримати.

Переваги та недоліки:
+ Зменшує залежність між клієнтом та обробниками: клієнт не знає хто обробить запит, а обробники не знають структуру ланцюжка.
+ Реалізовує додаткову гнучкість в обробці запиту: легко добавити або вилучити з ланцюжка нові обробники.
– Запит може залишитися ніким не опрацьованим: запит не має вказаного обробника, тому може бути не опрацьованим.

Хід роботи

Реалізація патерну проєктування

Для реалізації Web Crawler використано патерн проєктування Chain of Responsibility, оскільки він забезпечує гнучку послідовну обробку веб–сторінок через ланцюжок обробників, кожен з яких відповідає за конкретну операцію очищення чи аналізу контенту. Це дозволяє модульно додавати, видаляти або змінювати порядок обробки сторінок без змін в основній логіці краулера.

Патерн Chain of Responsibility реалізовано у класах PageHandler, AbstractPageHandler та конкретних обробниках, які утворюють гнучку ланцюжок обробки HTML–контенту. Такий підхід є особливо ефективним у роботі веб–краулера, де необхідно послідовно застосовувати різні трансформації та аналіз завантаженого контенту.

```java
public abstract class AbstractPageHandler implements PageHandler {

    3 usages
    protected PageHandler nextHandler;

    2 usages  dariahorlach
    public AbstractPageHandler setNext(PageHandler nextHandler) {
        this.nextHandler = nextHandler;
        return this;
    }

    3 usages  dariahorlach
    protected void handleNext(PageContext context) {
        if (nextHandler != null) {
            nextHandler.handle(context);
        }
    }
}
```

Рис. 2 – Код класу AbstractPageHandler

```java
public class AdsCleanerHandler extends AbstractPageHandler {

    2 usages  dariahorlach
    @Override
    public void handle(PageContext context) {
        System.out.println("Видаляємо рекламні блоки");
        context.setHtmlContent(
                context.getHtmlContent().replaceAll( regex: "<div class=\"ads\">.*?</div>",
        );
        handleNext(context);
    }
}
```

Рис. 3 – Код класу AdsCleanerHandler

```java
public class KeywordSearchHandler extends AbstractPageHandler {

    2 usages  dariahorlach
    @Override
    public void handle(PageContext context) {
        System.out.println("Пошук ключового слова: " + context.getKeyword());

        if (context.getHtmlContent().contains(context.getKeyword())) {
            System.out.println("Ключове слово знайдено!");
        } else {
            System.out.println("Ключове слово не знайдено");
        }

        handleNext(context);
    }
}
```

Рис. 4 – Код класу KeywordSearchHandler

```java
public class PageContext {

    3 usages
    private String htmlContent;
    2 usages
    private final String keyword;

    1 usage  dariahorlach
    public PageContext(String htmlContent, String keyword) {
        this.htmlContent = htmlContent;
        this.keyword = keyword;
    }

    4 usages  dariahorlach
    public String getHtmlContent() { return htmlContent; }

    2 usages  dariahorlach
    public void setHtmlContent(String htmlContent) {
        this.htmlContent = htmlContent;
    }

    2 usages  dariahorlach
    public String getKeyword() { return keyword; }
}
```

Рис. 5 – Код класу PageContext

```java
public interface PageHandler {
    2 usages  3 implementations  dariahorlach
    void handle(PageContext context);
}
```

Рис. 6 – Код класу PageHandler

```java
public class ScriptCleanerHandler extends AbstractPageHandler {

    2 usages  dariahorlach
    @Override
    public void handle(PageContext context) {
        System.out.println("Видаляємо <script> теги");
        context.setHtmlContent(
                context.getHtmlContent().replaceAll( regex: "<script.*?</script>", replacement: "")
        );
        handleNext(context);
    }
}
```

Рис. 7 – Код класу ScriptCleanerHandler

Використання цього патерну надає:

1. Модульну обробку контенту: Кожен обробник відповідає за одну конкретну операцію, що дозволяє легко додавати нові типи обробки без змін в існуючих класах.
2. Гнучкість у побудові ланцюжків: Можливість динамічно створювати різні комбінації обробників залежно від потреб.
3. Принцип єдиного обов'язку: Кожен обробник має чітко визначену відповідальність (очищення скриптів, видалення реклами, пошук ключових слів), що спрощує тестування та супровід коду.
4. Автоматичну передачу обробки: Базовий клас AbstractPageHandler забезпечує автоматичну передачу контексту наступному обробнику через метод handleNext(), спрощуючи розширення ланцюжка.

У нашому випадку патерн Chain of Responsibility забезпечує ефективну pipeline–обробку веб–сторінок. Це дозволяє створювати складні сценарії обробки контенту, де кожен етап може бути легко налаштований, замінений або вимкнений без впливу на інші компоненти системи.

Структура патерну

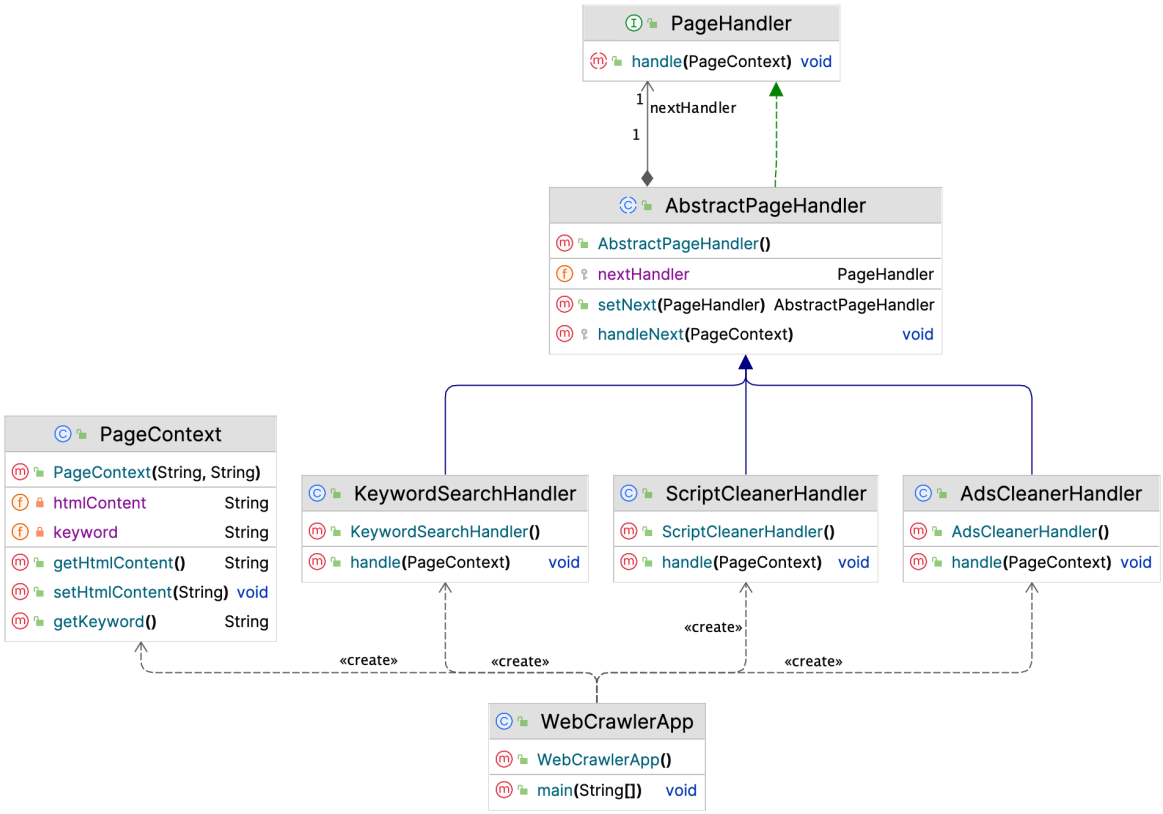


Рис. 8 – Структура патерну Ланцюжок відповідальності

Опис реалізації:

- PageHandler – інтерфейс, що визначає контракт для всіх обробників
- AbstractPageHandler – абстрактний базовий клас, що реалізує логіку зв'язування обробників
- ScriptCleanerHandler – видаляє JavaScript–код зі сторінки
- AdsCleanerHandler – очищає сторінку від рекламних блоків
- KeywordSearchHandler – шукає ключові слова в контенті
- PageContext – контейнер даних, що передається через ланцюжок обробки

Такий підхід особливо ефективний для веб–краулерів, де треба послідовно застосовувати різні фільтри до завантаженого контенту, керувати

порядком обробки залежно від специфіки завдань та адаптувати обробку під різні типи веб–сайтів.

Посилання на репозиторій: https://github.com/dariahorlach/Lab–TRPZ

Висновки

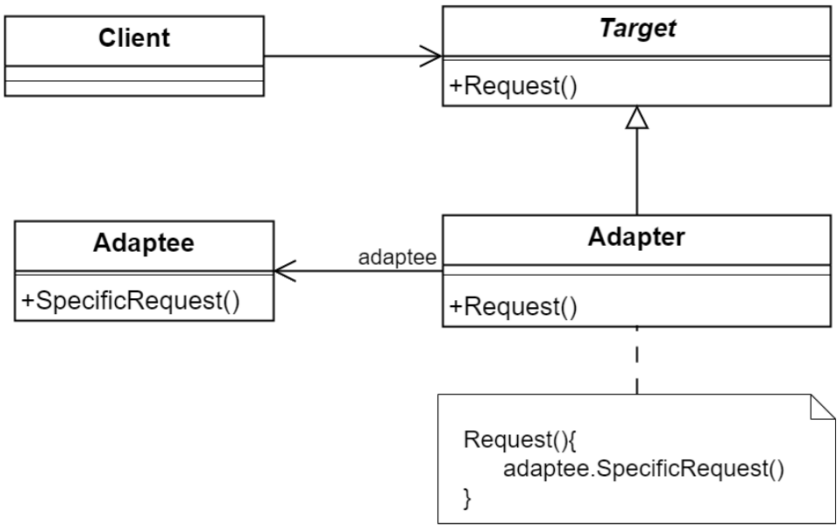
Висновки: під час виконання лабораторної роботи, було реалізовано патерн Chain of responsibility для веб–краулера, що дозволило створити гнучку модульну систему обробки веб-контенту через ланцюжок спеціалізованих обробників. Кожен обробник взяв на себе чітко визначену відповідальність, дотримуючись принципу єдиного обов'язку. Архітектура з Chain of Responsibility дозволила створити ефективну послідовність обробки, де завантажений через Proxy контент проходить через серію перетворень: очищення від JavaScript, видалення рекламних блоків та пошук ключових слів. Клас AbstractPageHandler забезпечив зручний механізм зв'язування обробників та автоматичної передачі PageContext через ланцюжок.

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).

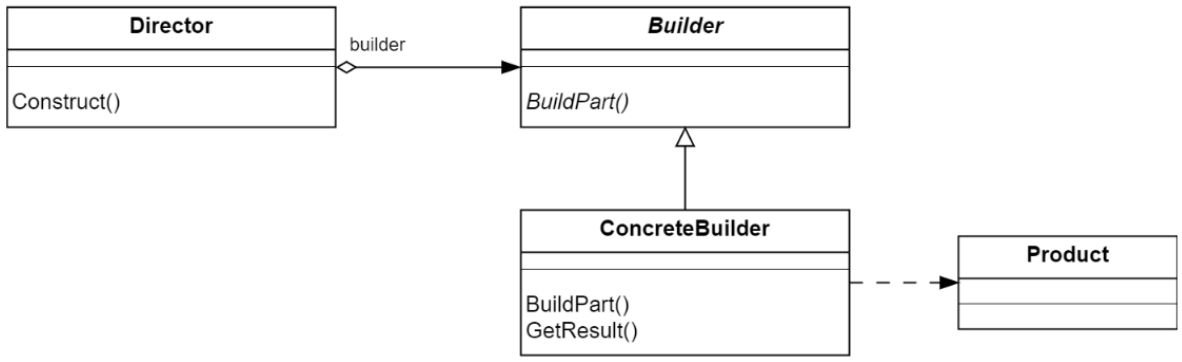
На рівні класів адаптер наслідує як Target, так і Adaptee (через множинне успадкування).

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.

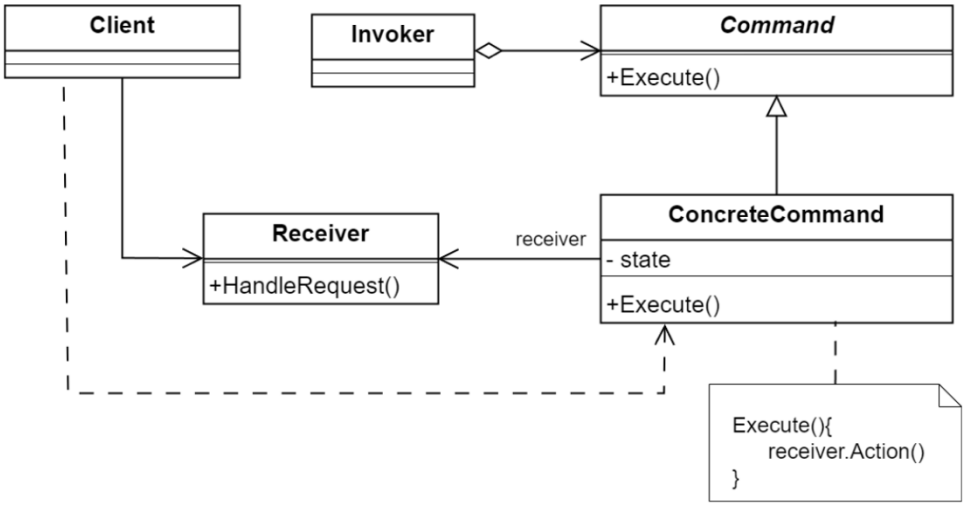
Коли потрібно створювати різні представлення одного об’єкта.

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.

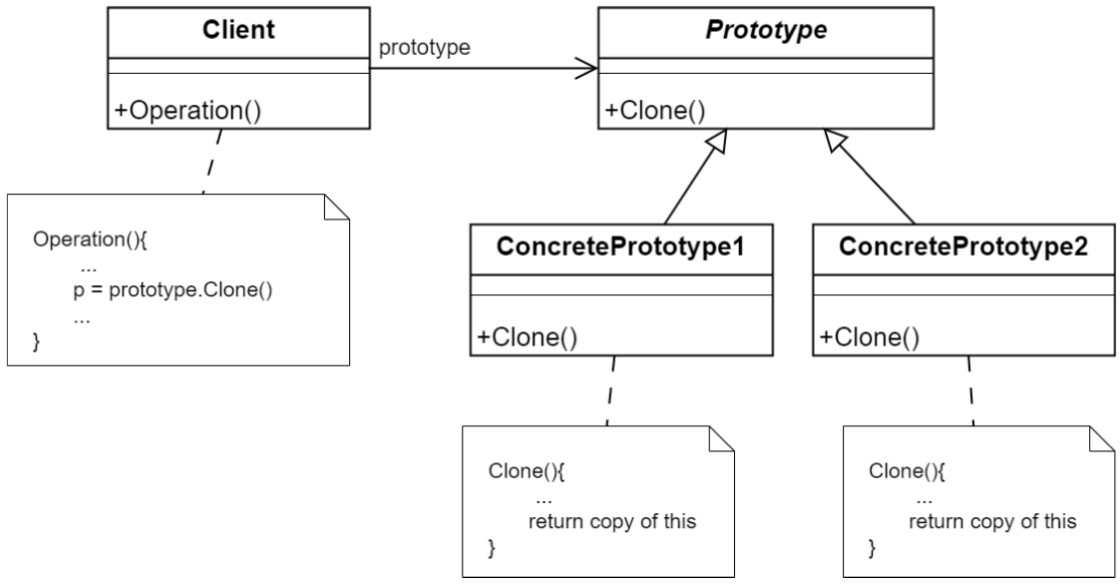
4. Команда виконує дію, викликаючи методи Receiver.

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.