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

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Тема роботи: 7. Редактор зображень

Виконав
студент групи ІА–33
Марченко Вадим Олександрович

Київ 2025

Тема: Патерни проектування

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

Посилання на репозиторій: https://github.com/nwu1015/ImageEditor

Короткі теоретичні відомості

Адаптер (Adapter)

Шаблон проєктування "Адаптер" дозволяє об’єднати інтерфейси двох несумісних класів. Це корисно, коли потрібно використовувати наявний клас, але його інтерфейс не відповідає потребам клієнтського коду. Адаптер огортає клас-джерело і надає інтерфейс, що сумісний з інтерфейсом, який очікує клієнт. Таким чином, клієнтський код може працювати з адаптером, не змінюючи свій інтерфейс. У Java цей шаблон часто реалізується шляхом створення нового класу, який наслідує або реалізує необхідний інтерфейс і всередині використовує методи класу-джерела. Шаблон дозволяє покращити модульність і повторне використання коду, зберігаючи при цьому стабільний інтерфейс для клієнта.

Будівельник (Builder)

Шаблон "Будівник" забезпечує гнучкий спосіб створення складних об’єктів, дозволяючи будувати їх покроково. Це особливо корисно, коли об'єкт має багато опціональних параметрів або складну структуру. Замість того щоб створювати об'єкт через конструктор з численними параметрами, будівник дає можливість додавати значення по черзі і в кінці отримати готовий об’єкт. У Java це часто реалізується за допомогою вкладеного статичного класу "Builder" всередині основного класу, де кожен метод будівника повертає об'єкт будівника, дозволяючи ланцюжковий виклик

методів. Така реалізація робить код читабельнішим і гнучкішим для створення об'єктів з різними конфігураціями.

Команда (Command)

Шаблон "Команда" інкапсулює дію або запит як об’єкт, дозволяючи відкладене виконання операції, її скасування або збереження для подальшого використання. Кожна команда реалізує інтерфейс із методами для виконання дії (наприклад, execute). Це дозволяє відокремити клієнтський код від отримувача дії. Команди можна комбінувати, записувати в лог або ставити в чергу. У Java цей шаблон можна реалізувати за допомогою інтерфейсу Command і його конкретних реалізацій для різних операцій. Це полегшує обробку запитів в інтерфейсі, де користувач може обирати команди, а також зручно для реалізації таких функцій, як "Скасувати" або "Повторити" в додатках

Ланцюг відповідальностей (Chain of Responsibility)

Шаблон "Ланцюг відповідальностей" дозволяє передавати запит вздовж ланцюга обробників, де кожен обробник має шанс обробити запит або передати його далі. Це дозволяє клієнтському коду не знати, хто саме оброблятиме запит, і забезпечує гнучкість в додаванні чи видаленні обробників. Кожен обробник реалізує інтерфейс із методом для обробки запиту і зберігає посилання на наступного обробника в ланцюзі. У Java цей шаблон можна реалізувати шляхом створення абстрактного класу для обробників, де кожен конкретний обробник перевіряє, чи здатен він обробити запит, або передає його далі. Це зручно для систем обробки помилок або запитів з різними рівнями доступу.

Прототип (Prototype)

Шаблон "Прототип" дозволяє створювати нові об'єкти шляхом копіювання вже наявного об'єкта (прототипу) замість створення об’єктів "з нуля". Це ефективно, коли створення нового об'єкта є складним або ресурсозатратним, а також корисно для створення об'єктів зі схожими початковими станами. У Java шаблон може бути реалізований через інтерфейс Cloneable, що вимагає реалізації методу clone, який повертає копію об’єкта. Завдяки цьому шаблону можна створювати об'єкти зі схожою конфігурацією без необхідності повторювати весь процес ініціалізації, зберігаючи при цьому незалежність їхніх станів.

Хід роботи

У цій роботі я використовую патерн "Прототип" (Prototype). Цей патерн було обрано як архітектурне рішення для спрощення створення нових об’єктів ImageLayer на основі вже існуючих.

У процесі роботи над проєктом стало зрозуміло, що при редагуванні колажу користувач може захотіти скопіювати один із шарів (наприклад, з тими ж параметрами розміру, позиції, обертання чи ефектів). Замість того щоб створювати новий об’єкт вручну й повторно задавати всі параметри, значно ефективніше просто клонувати вже готовий шар.

Без використання патерну Prototype, така логіка потребувала б дублювання коду: у кожному місці, де потрібно створити копію шару, довелося б вручну копіювати всі його властивості. Це не лише ускладнило б підтримку коду, а й збільшило ризик появи помилок, якщо згодом у модель додавалися нові поля.

Патерн Prototype вирішує цю проблему елегантно. Він дозволяє створювати нові об’єкти, копіюючи існуючі, без необхідності залежати від їх конкретного класу чи конструктора. У моєму проєкті цей підхід реалізований через метод clone() у класі ImageLayer, який створює точну

копію шару, зберігаючи його властивості (позицію, розмір, ефекти тощо), але водночас дозволяє змінювати унікальні параметри — наприклад, z-Index.

Реалізація шаблону “State” вимагає заповнення методу clone() у сутності для реалізації певної поведінки в залежності від стану об’єкту:

```
3   public interface Prototype<T> {  2 usages  1 implementation  nwu1015
4       T clone();  1 implementation  nwu1015
5   }
```

Рисунок 1. - Інтерфейс з методом clone() для класичного Prototype.

```
@Override  nwu1015 *
public ImageLayer clone() {
    try {
        ImageLayer newLayer = (ImageLayer) super.clone();

        newLayer.setId(null);

        newLayer.setCollage(this.getCollage());
        newLayer.setImage(this.getImage());

        newLayer.setWidth(this.width);
        newLayer.setHeight(this.height);
        newLayer.setRotationAngle(this.rotationAngle);
        newLayer.setCropX(this.cropX);
        newLayer.setCropY(this.cropY);
        newLayer.setCropWidth(this.cropWidth);
        newLayer.setCropHeight(this.cropHeight);
        newLayer.setPositionX(this.positionX);
        newLayer.setPositionY(this.positionY);

        return newLayer;
    }catch (CloneNotSupportedException e) {
        throw new RuntimeException("Something went wrong.. Can't clone layer.");
    }
}
```

Рисунок 2. - Реалізація шаблону Prototype для класу ImageLayer.

```
@Transactional  1 usage   nwu1015 *
public void duplicateLayer(Long layerId) {
    Prototype<ImageLayer> prototypeLayer = imageLayerRepository.findById(layerId)
            .orElseThrow(() -> new RuntimeException("Layer not found: " + layerId));

    Collage collage = ((ImageLayer) prototypeLayer).getCollage();
    collage.getCurrentState().checkCanEdit(collage);

    ImageLayer newLayer = prototypeLayer.clone();

    int maxZIndex = collage.getLayers().stream()  Stream<ImageLayer>
            .mapToInt(ImageLayer::getZIndex)  IntStream
            .max().orElse( other: -1);
    newLayer.setZIndex(maxZIndex + 1);

    imageLayerRepository.save(newLayer);
}
```

Рисунок 3. - Використання шаблону при клонуванні шару при створенні колажу.

У класі ImageLayer реалізовано метод clone(), який відповідає за створення копії об’єкта поточного шару зображення. Метод використовується для реалізації патерну Prototype, що дозволяє створювати нові об’єкти шляхом копіювання вже існуючих, замість створення їх "з нуля".

Метод виконує поверхневе копіювання об’єкта, викликаючи super.clone(), після чого створює новий екземпляр класу ImageLayer, який має такі ж значення полів, як і оригінал. Ідентифікатор (id) при цьому обнуляється, щоб новий об’єкт можна було зберегти як окремий запис у базі даних.

Далі копіюються всі властивості шару – посилання на об’єкти Collage і Image, а також параметри позиції, розмірів, повороту, кадрування та ефектів. У випадку, якщо клонування не підтримується (тобто виникає виняток CloneNotSupportedException), метод перехоплює цю помилку та генерує RuntimeException.

Діаграма класів:

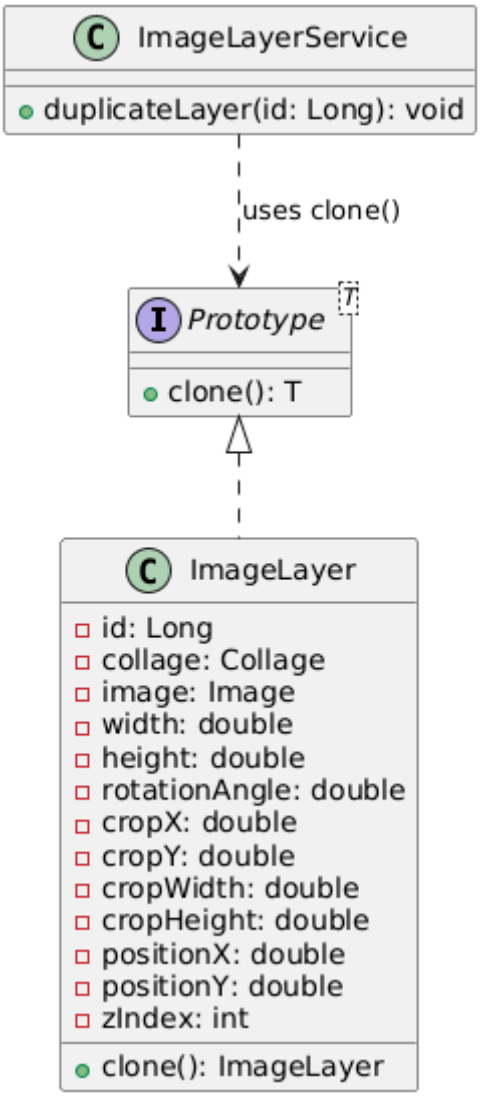


Рисунок 4. - Діаграма класів використання шаблону Prototype

Висновок: Виконуючи цю лабораторну роботу, я ознайомився з такими патернами, як Adapter, Builder, Command, Chain of Responsibility, Prototype.

Особливу увагу я приділив реалізації шаблону Prototype, детально описавши його основну логіку та функціональність у контексті мого проєкту. Цей патерн було обрано як архітектурне рішення для спрощення створення нових об’єктів ImageLayer на основі вже існуючих.

Під час реалізації шаблону Prototype я зрозумів, наскільки цей патерн спрощує процес дублювання складних об’єктів, особливо тих, що мають

численні параметри та зв’язки з іншими сутностями. Це робить код більш гнучким, зручним для розширення та повторного використання. Крім того, використання методу clone() допомагає зменшити кількість помилок, пов’язаних із ручним копіюванням даних, і забезпечує більш чисту та структуровану архітектуру програми.

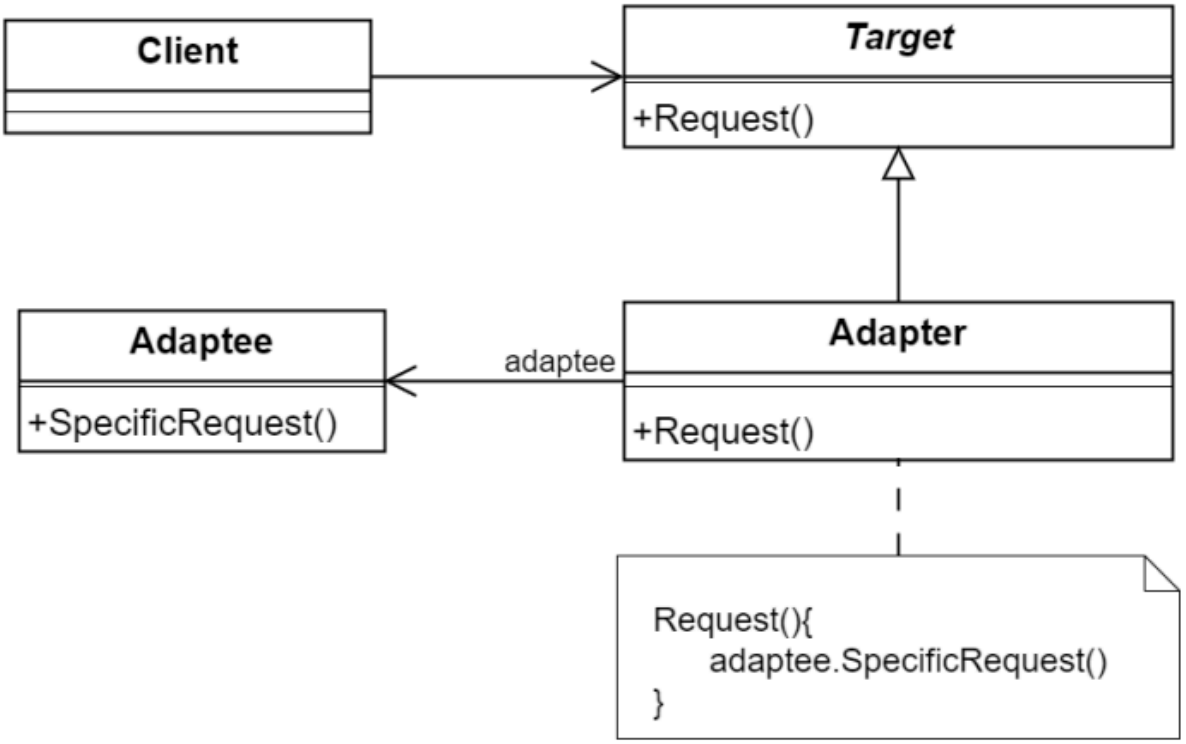
Завдяки цьому досвіду я краще зрозумів переваги патерну Prototype у контексті роботи з об’єктами, які часто потребують створення подібних копій, а також його роль у підвищенні ефективності та узгодженості програмного коду.

Контрольні запитання

1. Яке призначення шаблону «Адаптер»?

Призначення шаблону «Адаптер» - забезпечити спільну роботу класів з несумісними інтерфейсами. Він діє як "перехідник" або "обгортка" навколо існуючого класу (Adaptee), перетворюючи його інтерфейс на інший, очікуваний клієнтським кодом (Client)

2. Нарисуйте структуру шаблону «Адаптер».



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

Client (Клієнт): Клас, який хоче використовувати функціонал, але очікує певний інтерфейс (Target). У нашій роботі це RarStrategy. Target (Цільовий інтерфейс): Інтерфейс, який використовує Client. Adapter (Адаптер): Клас, який реалізує інтерфейс Target і містить посилання на об'єкт Adaptee. Він перенаправляє виклики від Client до Adaptee, виконуючи необхідні перетворення. У нашій роботі це RarAdapter. Adaptee (Об'єкт, що адаптується): Існуючий клас з несумісним інтерфейсом. У нашій роботі це LegacyRarEngine.

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

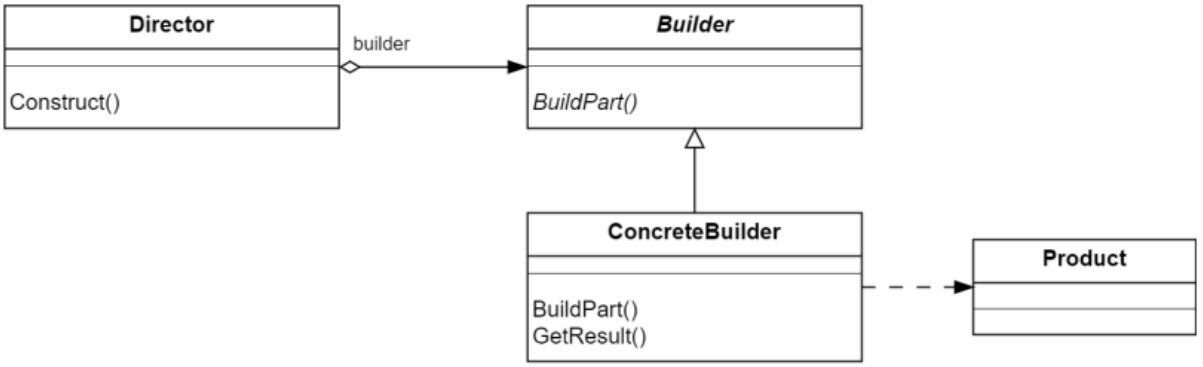
Адаптер об'єктів (використано в роботі): Використовує композицію. Клас Adapter містить екземпляр класу Adaptee. Цей підхід є більш гнучким, оскільки дозволяє адаптувати будь-який підклас Adaptee.

Адаптер класів: Використовує множинне успадкування (в Java реалізується через успадкування класу та реалізацію інтерфейсу). Клас Adapter одночасно успадковує Adaptee та реалізує інтерфейс Target. Цей підхід менш гнучкий.

5. Яке призначення шаблону «Будівельник»?

Шаблон «Будівельник» (Builder) використовується для покрокового створення складних об'єктів. Він дозволяє відокремити процес конструювання об'єкта від його представлення, завдяки чому один і той самий процес конструювання може створювати різні представлення

6. Нарисуйте структуру шаблону «Будівельник».



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

Product (Продукт): Складний об'єкт, який створюється. Builder (Будівельник): Абстрактний інтерфейс для створення частин об'єкта Product. ConcreteBuilder (Конкретний будівельник): Реалізує інтерфейс Builder і конструює конкретне представлення продукту. Director (Директор): Клас, який керує процесом побудови, використовуючи об'єкт Builder.

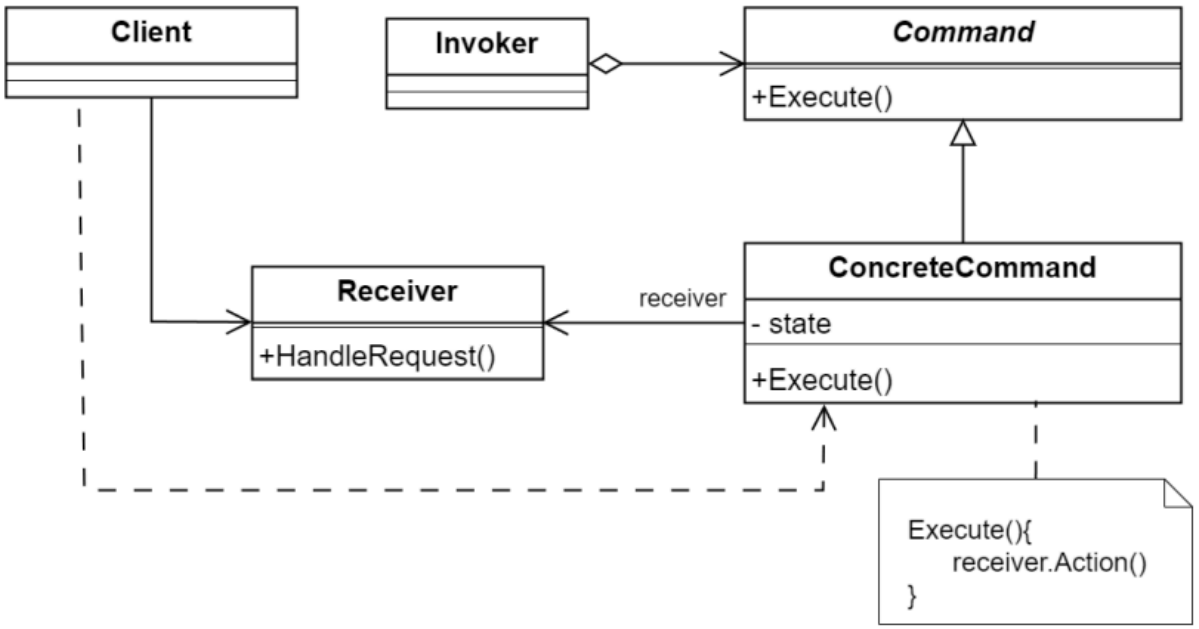
8. У яких випадках варто застосовувати шаблон «Будівельник»"?

Коли процес створення об'єкта є складним, багатоетапним, або коли конструктор має занадто багато параметрів (особливо опціональних). Також, коли потрібно створювати різні представлення одного й того ж об'єкта.

9. Яке призначення шаблону «Команда»?

Шаблон «Команда» (Command) інкапсулює запит на виконання дії як об'єкт. Це дозволяє параметризувати клієнтські об'єкти різними запитами, ставити запити в чергу, логувати їх, а також підтримувати операції скасування (undo).

10. Нарисуйте структуру шаблону «Команда».



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

Command: Інтерфейс, що оголошує метод для виконання операції (execute). ConcreteCommand: Реалізує інтерфейс Command, містить посилання на Receiver і викликає його методи. Client: Створює об'єкт

ConcreteCommand і встановлює його одержувача. Invoker: Просить команду виконати запит. Receiver: "Одержувач", який знає, як виконати операцію.

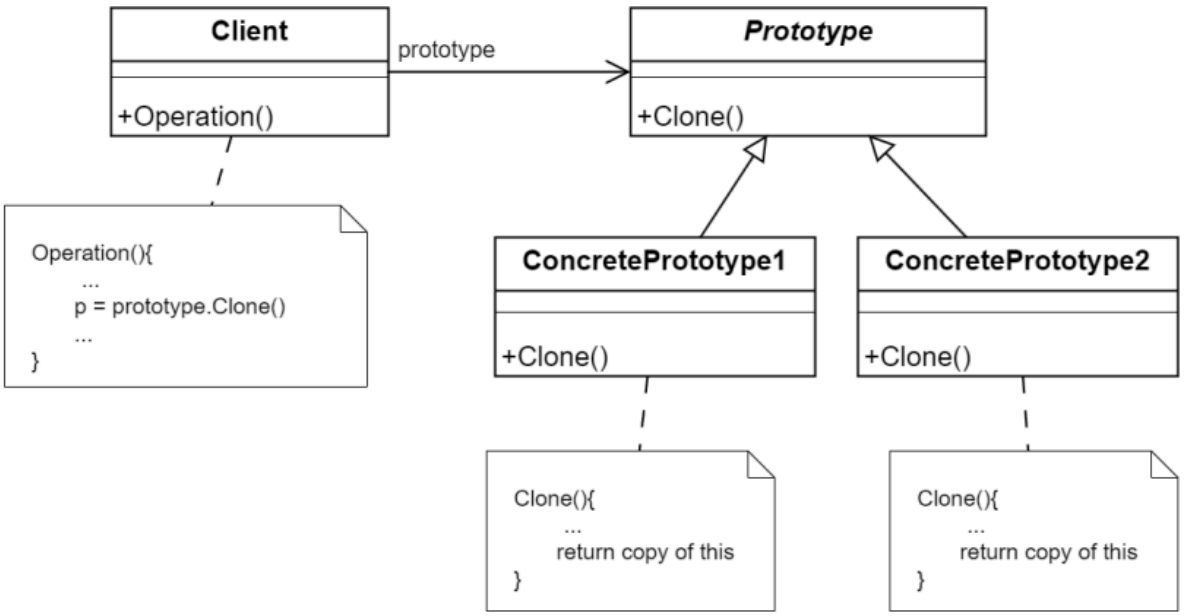
12. Розкажіть як працює шаблон «Команда».

Клієнт створює об'єкт-команду, пов'язуючи її з конкретним одержувачем. Потім цей об'єкт-команда передається ініціатору (Invoker), наприклад, кнопці в меню. Коли користувач натискає кнопку, ініціатор викликає метод execute() у команди, а команда, в свою чергу, викликає потрібний метод у свого одержувача.

13. Яке призначення шаблону «Прототип»?

Шаблон «Прототип» (Prototype) дозволяє створювати нові об'єкти шляхом копіювання існуючого об'єкта (прототипу). Це дозволяє уникнути прив'язки до класів об'єктів, що створюються.

14. Нарисуйте структуру шаблону «Прототип».



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

Prototype: Інтерфейс, що оголошує метод клонування (clone). ConcretePrototype: Реалізує інтерфейс Prototype та метод clone. Client: Створює новий об'єкт, викликаючи метод clone у прототипу.

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

Обробка подій в GUI: Коли користувач клікає на кнопку, подія спочатку обробляється кнопкою, потім може бути передана батьківській панелі, потім вікну, і так далі, доки не буде оброблена. Системи логування: Повідомлення може проходити через ланцюжок обробників: один записує в консоль, інший - у файл, третій - відправляє по email, залежно від рівня важливості.

Системи авторизації та валідації: Запит користувача може проходити через ланцюжок перевірок: перевірка автентифікації, перевірка прав доступу, перевірка валідності даних.