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 №7
Технології розроблення програмного забезпечення
«Патерни проектування»

Тема: Редактор зображень

Виконала:

студентка групи ІА-32

Ейсмонт А.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

Тема: Патерни проектування.

Мета: Вивчити структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

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

Теоретичні відомості:

Шаблон «Facade»

Призначення патерну: Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми [6]. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагетікоду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей. Це також відволікає користувачів від змін в підсистемі (внутрішня реалізація може змінюватися, а наданої послуги немає), що також скоротить кількість змін в використовуваних фасад класах (без фасаду довелося б змінювати вихідні коди в безлічі точок). Звичайно, твердої умови повного закриття внутрішніх класів підсистеми не стоїть – при необхідності можна звертатися до окремих класів безпосередньо, минаючи об'єкт фасад.

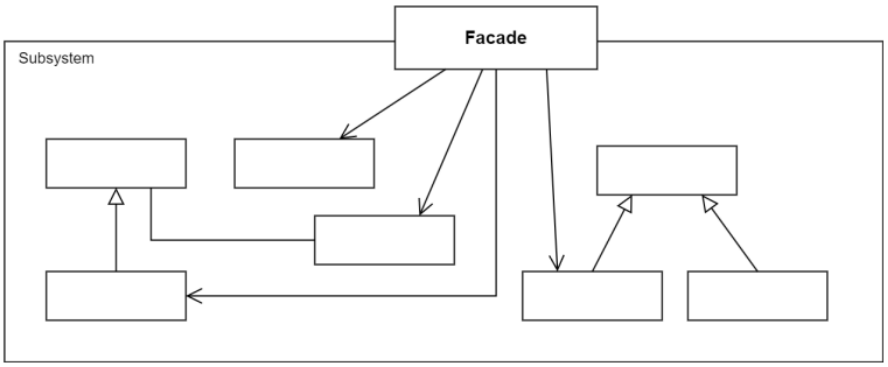
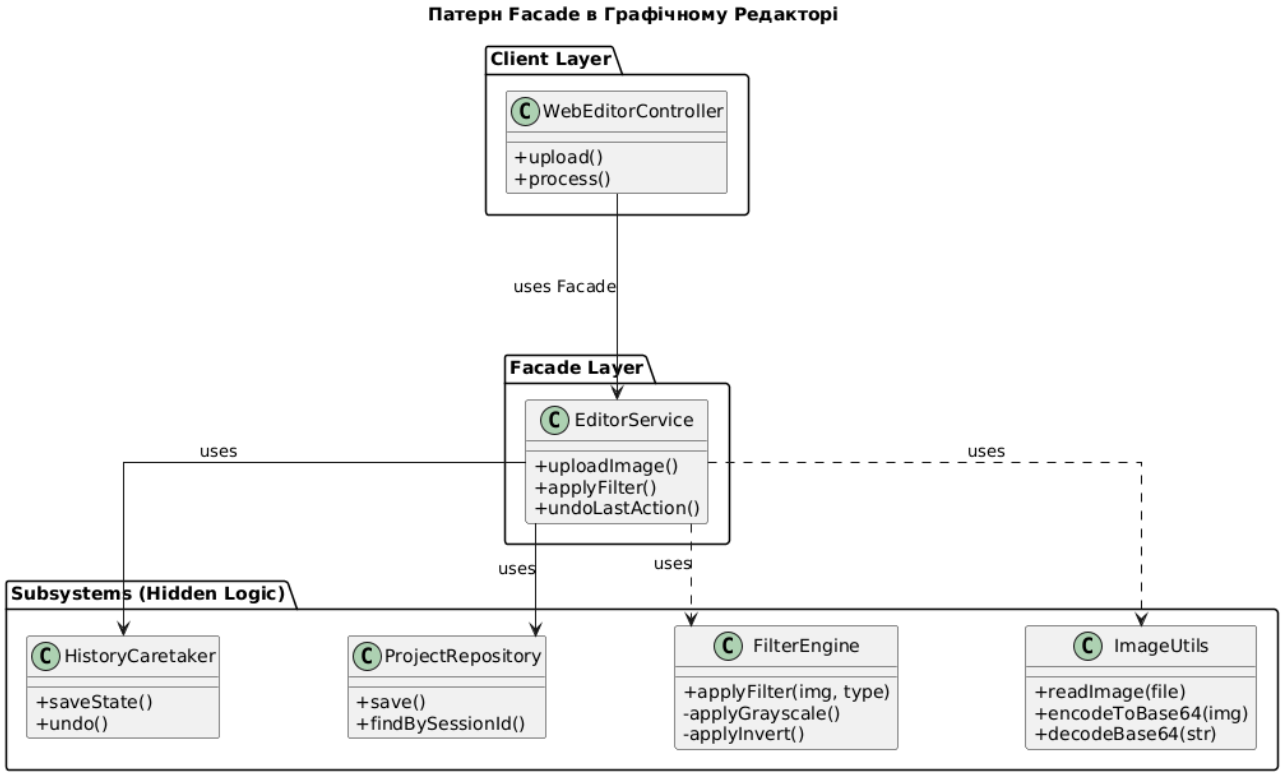


Рисунок 7.2. Структура патерну «Фасад»

Проблема: Ви розробляєте компонент, який дозволяє відправляти запити на різні типи endpoint, а також працює з протоколами HTTP та TCP/IP. Прототип компонента вже працює, але структура класів вийшла досить складна, а при налаштуванні на різні протоколи мають використовуватися різні класи. Інструкція для використання також виходить досить складна та заплутана. Слід добавити, так як інші системи будуть знати про внутрішню будову вашого компонента, а тому при спробі змінити внутрішню структуру в наступних версіях, вам прийдеться повідомляти про це всіх користувачів вашого компонента і для них перехід на наступну версію вашого компонента буде достатньо складним. Рішення: Тут краще використати патерн фасад. А саме, в даному випадку, створити один клас, наприклад, InternetClient, та набір методів у нього, які будуть використовуватися для налаштування цього підключення, та його подальшого використання. Цей клас єдиний буде позначено як public, і тільки його будуть бачити ваші клієнти. Таким чином, з точки зору зовнішнього коду вони будуть працювати з набагато простішим інтерфейсом, а значить і інструкція використання буде значно простіша. Крім того, так як внутрішня структура повністю закрита, то в наступних версіях ви можете її змінювати, як вам буде зручно, а з точки зору користувачів компонента, перехід на нову версію буде просто переключенням на нову версію бібліотеки. Переваги та недоліки: + Інкапсуляція внутрішньої структури від клієнтського коду. + Спрощується інтерфейс для роботи з модулем закритим фасадом. - Зниження гнучкості в налаштуванні та використанні програмного коду закритого фасадом.

Хід роботи:

Діаграма класів:



Опис реалізації патерну «Facade»

У цій лабораторній роботі було реалізовано структурний патерн проектування Facade (Фасад). Його метою є приховування складності системи шляхом надання спрощеного інтерфейсу до набору інтерфейсів підсистем.

До впровадження патерну клас EditorService був перевантажений різнорідною логікою: робота з базою даних, обробка графіки (піксельні операції), конвертація форматів файлів (Base64/Bytes) та керування історією.

Було проведено рефакторинг та виділено наступні підсистеми:

- **ImageUtils** (Subsystem): Відповідає за низькорівневі операції введення-виведення, конвертацію між MultipartFile, BufferedImage та Base64 рядками.
- **FilterEngine** (Subsystem): Містить алгоритми обробки зображень (градації сірого, інверсія).
- **ProjectRepository** та HistoryCaretaker (Subsystem): Відповідають за збереження даних та історії.

Клас **EditorService** тепер виступає в ролі Фасаду. Клієнт (контролер) звертається до нього через прості методи (uploadImage, applyFilter), а Фасад делегує виконання роботи відповідним класам підсистеми. Це зменшило зв'язність коду (loose coupling) та спростило його підтримку.

Фрагменти програмного коду:

– Клас підсистеми (**ImageUtils**) — конвертація форматів та робота з потоками введення-виведення.:

```
@Component
public class ImageUtils {
    public BufferedImage readImage(MultipartFile file) throws IOException {
        return ImageIO.read(new ByteArrayInputStream(file.getBytes()));
    }

    public String encodeToBase64(BufferedImage image) {
        try (ByteArrayOutputStream os = new ByteArrayOutputStream()) {
            ImageIO.write(image, "png", os);
            return Base64.getEncoder().encodeToString(os.toByteArray());
        } catch (IOException e) {
            throw new RuntimeException("Помилка кодування", e);
        }
    }

    // ... метод decodeBase64
}
```

– Клас підсистеми (**FilterEngine**) — алгоритми обробки зображень.:

```
@Component
public class FilterEngine {
    public void applyFilter(BufferedImage img, String filterType) {
        switch (filterType) {
            case "grayscale": applyGrayscale(img); break;
```

```
            case "invert": applyInvert(img); break;
        }
    }
    // Реалізація алгоритмів
}
```

– Клас підсистеми (**HistoryCaretaker**) — керування стеком історії змін.:

```
@Component
public class HistoryCaretaker {

    public void saveState(String sessionId, ProjectMemento memento) {
        // ... логіка збереження
    }

    public ProjectMemento undo(String sessionId) {
        // ... логіка відновлення
    }
}
```

– Клас Фасаду (**EditorService**):

```
@Service
public class EditorService {
    // Посилання на підсистеми
    private final ImageUtils imageUtils;
    private final FilterEngine filterEngine;
    // ...

    public String applyFilter(String sessionId, String type) {
        // 1. Отримання даних (через репозиторій)
        Project project = getProject(sessionId);
```

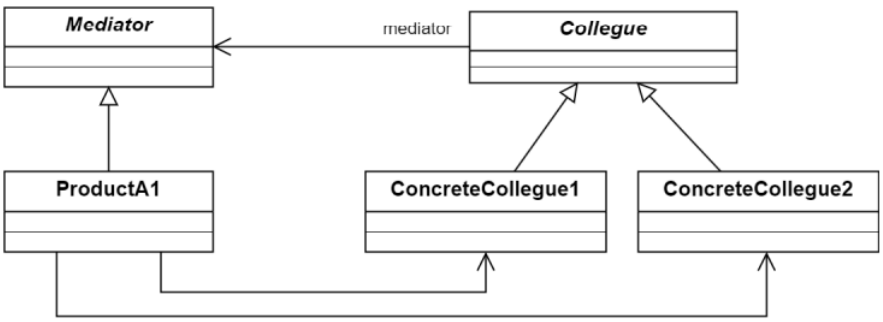
```
        BufferedImage bg = project.getLayer(0).getImage();

        // 2. Делегування складної роботи підсистемі
        filterEngine.applyFilter(bg, type);

        // 3. Конвертація результату через іншу підсистему
        return imageUtils.encodeToBase64(bg);
    }
}
```

Питання до лабораторної роботи:

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» призначений для визначення взаємодії об'єктів за допомогою іншого спеціального об'єкта, замість того, щоб об'єкти зберігали посилання один на одного. Цей шаблон зручно застосовувати, коли безліч об'єктів взаємодіють між собою складним чином, дозволяючи винести логіку взаємодії в окремий об'єкт .

2. Нарисуйте структуру шаблону «Посередник».



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

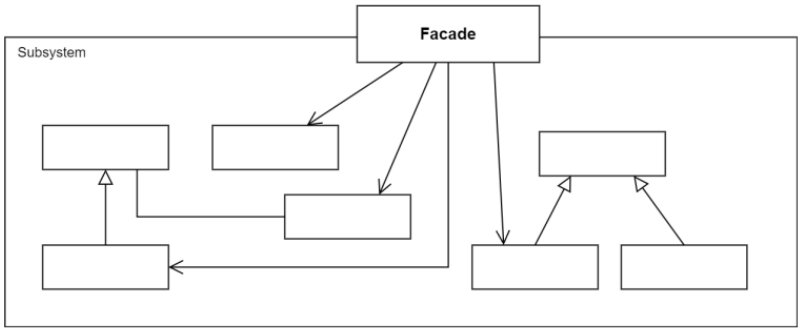
- Mediator (Посередник): визначає інтерфейс для обміну інформацією з об'єктами-колегами.
- Colleague (Колега): класи, що взаємодіють між собою.

Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор». Замість того щоб звертатися один до одного напряму, об'єкти звертаються до посередника, який перенаправляє запити або керує логікою взаємодії, нагадуючи диригента в оркестрі.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття її внутрішніх деталей. Він дозволяє уникнути тісної пов'язаності («спагеті-коду») і надає загальний інтерфейс доступу

5. Нарисуйте структуру шаблону «Фасад».



6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

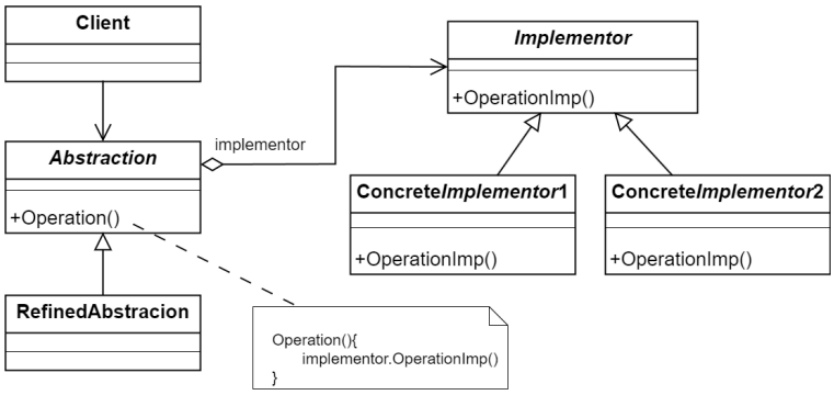
Facade (Фасад) та класи підсистеми (Subsystem classes).

Клієнти звертаються до Фасаду через його уніфікований інтерфейс. Фасад знає, яким класам підсистеми перенаправити запит, і викликає відповідні методи внутрішніх класів. Це ізолює клієнтів від змін у внутрішній реалізації підсистеми.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» використовується для поділу інтерфейсу (абстракції) і його реалізації, щоб вони могли змінюватися незалежно один від одного. Це необхідно, коли існує кілька різних абстракцій, над якими можна проводити дії різними способами.

8. Нарисуйте структуру шаблону «Міст».



9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

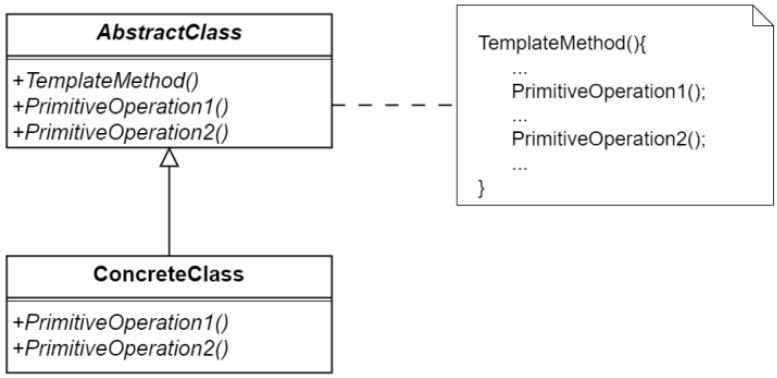
- *Abstraction (Абстракція) та RefinedAbstraction (Уточнена абстракція).*
- *Implementor (Реалізатор) та ConcreteImplementor (Конкретний реалізатор).*

Клас Абстракції містить агрегацію (посилання) на об'єкт Реалізатора. Абстракція делегує виконання низькорівневих операцій об'єкту-реалізатору.

10.Яке призначення шаблону «Шаблоний метод»?

Цей шаблон дозволяє реалізувати алгоритм покроково в абстрактному класі, залишаючи реалізацію специфічних кроків підкласам.

11.Нарисуйте структуру шаблону «Шаблоний метод».



12.Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

AbstractClass (базовий клас) та ConcreteClass (дочірній клас).

У базовому класі визначено метод, який містить загальний алгоритм (скелет). Цей метод викликає абстрактні або віртуальні методи, які мають бути реалізовані або перевизначені в дочірніх класах . Таким чином, загальна логіка залишається в базовому класі, а специфічна — в дочірніх.

13.Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Область використання «Шаблонного методу» — це покрокове визначення конкретного алгоритму, і він не обов'язково створює нові об'єкти, а лише визначає послідовність дій . «Фабричний метод» (або віртуальний конструктор) націлений саме на створення об'єктів і делегування процесу інстанціювання підкласам.

14.Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає функціональність розділення абстракції та реалізації, що дозволяє змінювати їх ієрархії незалежно одна від одної. Це дозволяє уникнути комбінаторного вибуху кількості класів при додаванні нових типів абстракцій або нових реалізацій.

Вихідний код:
https://github.com/eismonta/TRPZ_2025/tree/800fba78e5bb331df3f4d7b874b85e930e16525b/image-editor

Висновок:

Під час виконання лабораторної роботи я ознайомилась з структурою шаблонів «Mediator», «Facade», «Bridge», «Template method» та застосувала структурний патерн Facade (Фасад) для покращення архітектури свого проекту.