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

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Патерни проектування»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю.

Київ 2025

Тема: Патерни проектування

Тема проєкту: 28. JSON Tool (ENG) (strategy, command, observer, template method,flyweight) Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Autosave\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

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

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1) Ознайомитись з короткими теоретичними відомостями

Шаблон «Mediator»

Призначення патерну: Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного) [6]. Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами.

Даний шаблон зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння. У такому випадку вся логіка взаємодії виноситься в окремий об'єкт. Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор». «Медіатор» нагадує диригента при управлінні оркестром. Диригент стежить за тим, щоб кожен інструмент грав в правильний час і в злагоді з іншими інструментами. Функції «медіатора» повністю це повторюють.

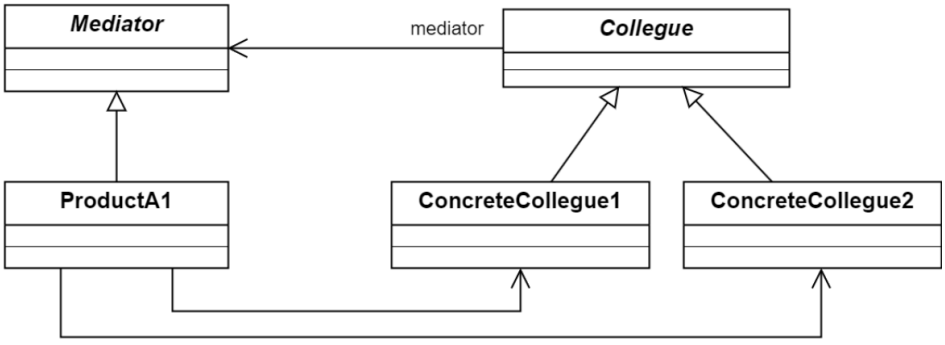


Рисунок 7.1. Структура патерна «Медіатор»

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.

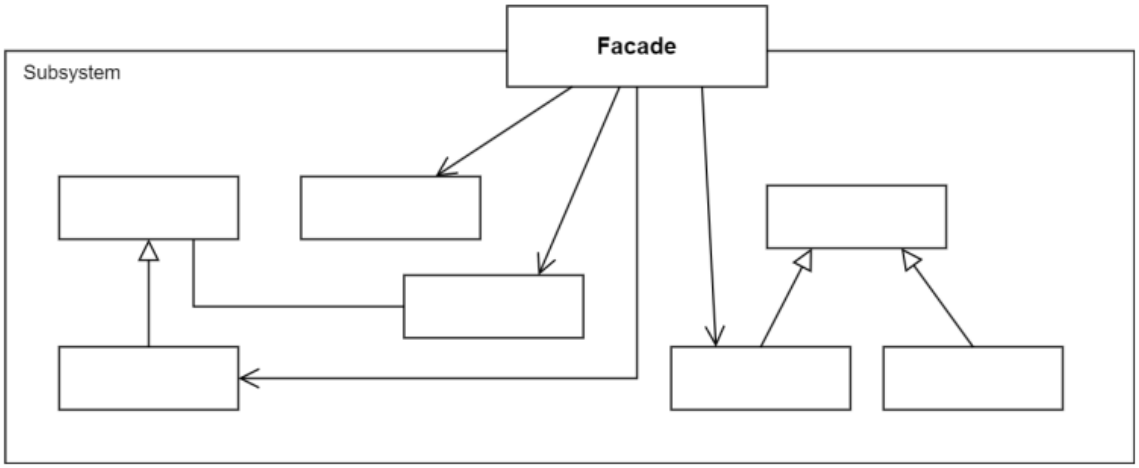


Рисунок 7.2. Структура патерну «Фасад»

Шаблон «Bridge» Призначення патерну: Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

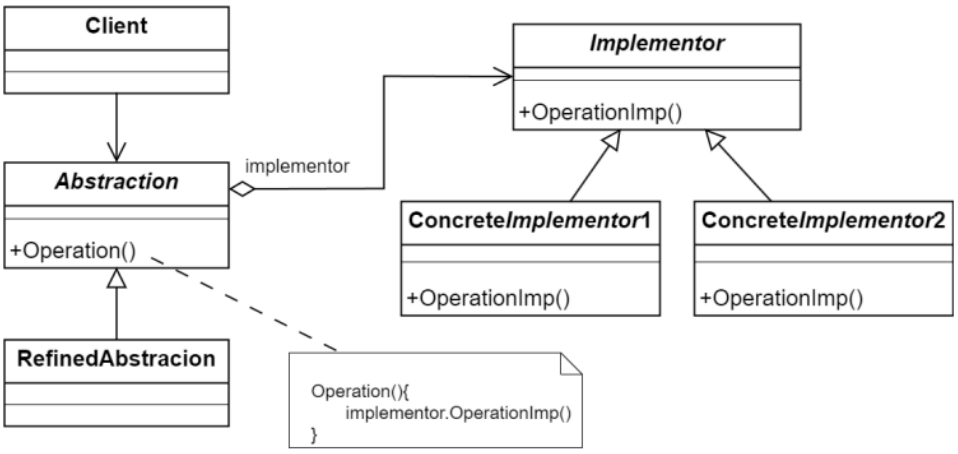


Рисунок 7.3. Структура патерну «Міст»

Шаблон «Template Method» Призначення патерну: Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам [6]. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом.

Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти – лише визначає послідовність дій.

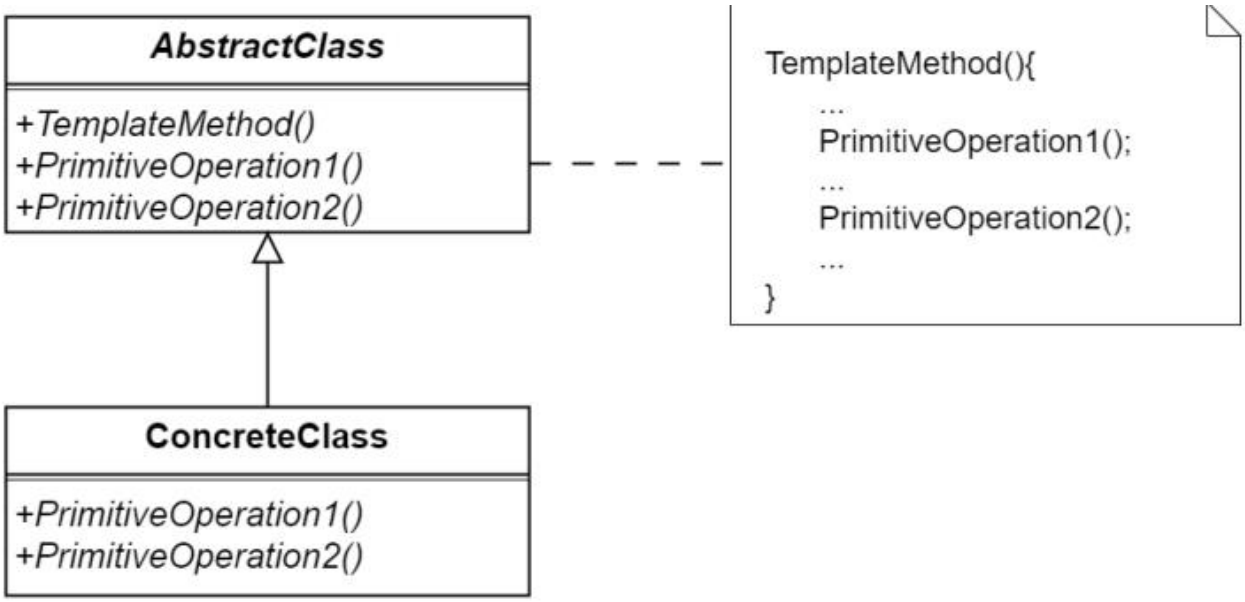


Рисунок 7.4. Структура патерну «Шаблонний метод»

2) Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3) Реалізувати один з розглянутих шаблонів за обраною темою.
4) Реалізувати не менше 3-х класів відповідно до обраної теми.

В даній частині було реалізовану логіку додавання файлів в форматі Markdown та .txt.

```java
package com.example.jsontool.template;

import com.example.jsontool.model.JSONDocument;


public abstract class DocumentExporter {  2 usages  2 inheritors  new *

    public final String exportDocument(JSONDocument document) {  2 usag
        StringBuilder sb = new StringBuilder();

        sb.append(generateHeader(document));
        sb.append("\n");
        sb.append(generateBody(document));
        sb.append("\n");
        sb.append(generateFooter(document));

        return sb.toString();
    }


    protected abstract String generateHeader(JSONDocument document);

    protected abstract String generateBody(JSONDocument document);  1 u

    protected abstract String generateFooter(JSONDocument document);
}
```

DocumentExporter.java

```java
package com.example.jsontool.template;

import com.example.jsontool.model.JSONDocument;
import org.springframework.stereotype.Component;

@Component  new *
public class MarkdownExporter extends DocumentExporter {

    @Override  1 usage  new *
    protected String generateHeader(JSONDocument document) {
        return "# Експорт Документа: " + document.getName();
    }

    @Override  1 usage  new *
    protected String generateBody(JSONDocument document) {
        return "## Зміст JSON\n```json\n" + document.getContent() + "\n```";
    }

    @Override  1 usage  new *
    protected String generateFooter(JSONDocument document) {
        return "---\n*Згенеровано автоматично JSON Tool*";
    }
}
```

MarkdownExporter.java

```java
package com.example.jsontool.template;

import com.example.jsontool.model.JSONDocument;
import org.springframework.stereotype.Component;

@Component  new *
public class TxtExporter extends DocumentExporter {

    @Override  1 usage  new *
    protected String generateHeader(JSONDocument document) {
        return "========================================\n" +
                "ДОКУМЕНТ: " + document.getName() + "\n" +
                "========================================";
    }

    @Override  1 usage  new *
    protected String generateBody(JSONDocument document) {
        return document.getContent();
    }

    @Override  1 usage  new *
    protected String generateFooter(JSONDocument document) {
        return "\n========================================\n" +
                "Кінець файлу.";
    }
}
```

TxtExporter.java

```java
    @Autowired
    private MarkdownExporter markdownExporter;

    @Autowired
    private TxtExporter txtExporter;

```

```java
    @GetMapping("/{id}/export/markdown") new *
    public ResponseEntity<String> exportToMarkdown(@PathVariable Long id) {
        JSONDocument document = documentService.findAll().stream() Stream<JSONDocument>
                .filter( JSONDocument d -> d.getId().equals(id))
                .findFirst() Optional<JSONDocument>
                .orElseThrow(() -> new IllegalArgumentException("Invalid document Id:" + id));

        String markdownContent = markdownExporter.exportDocument(document);

        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, ...headerValues: "attachment; filename=\"" + docume
                .contentType(MediaType.TEXT_MARKDOWN)
                .body(markdownContent);
    }

    @GetMapping("/{id}/export/txt") new *
    public ResponseEntity<String> exportToTxt(@PathVariable Long id) {
        JSONDocument document = documentService.findAll().stream() Stream<JSONDocument>
                .filter( JSONDocument d -> d.getId().equals(id))
                .findFirst() Optional<JSONDocument>
                .orElseThrow(() -> new IllegalArgumentException("Invalid document Id:" + id));

        String txtContent = txtExporter.exportDocument(document);

        return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, ...headerValues: "attachment; filename=\"" + docume
                .contentType(MediaType.TEXT_PLAIN)
                .body(txtContent);
    }

}
```

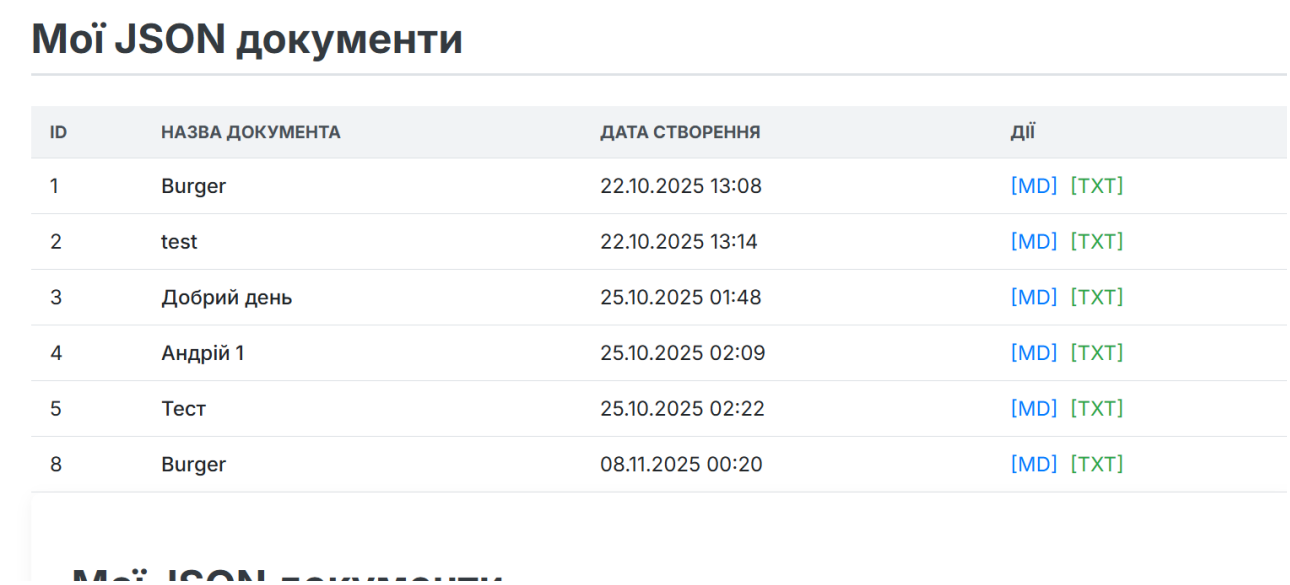
DocumentController.java

```html
            <td>
                <a th:href="@{/documents/{id}/export/markdown(id=${doc.id})}"
                   style="margin-right: 5px; text-decoration: none; color: #007bff;">
                    [MD]
                </a>
                <a th:href="@{/documents/{id}/export/txt(id=${doc.id})}"
                   style="text-decoration: none; color: #28a745;">
                    [TXT]
                </a>
            </td>
```

documents.html

Демонастрація роботи патерну

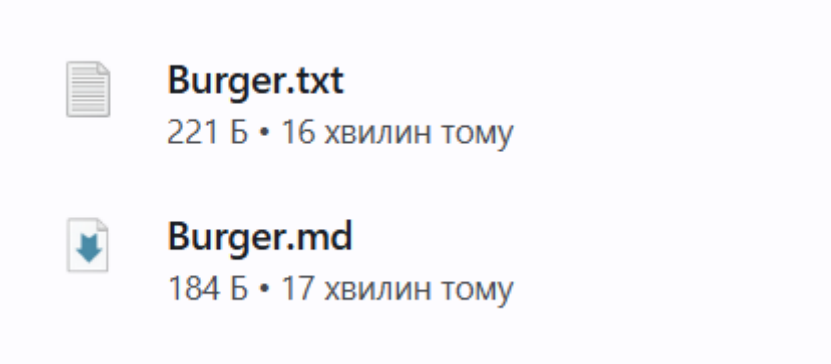
В цій лабораторній роботі ми виконали вимогу теми про експорт нашого json файлу.



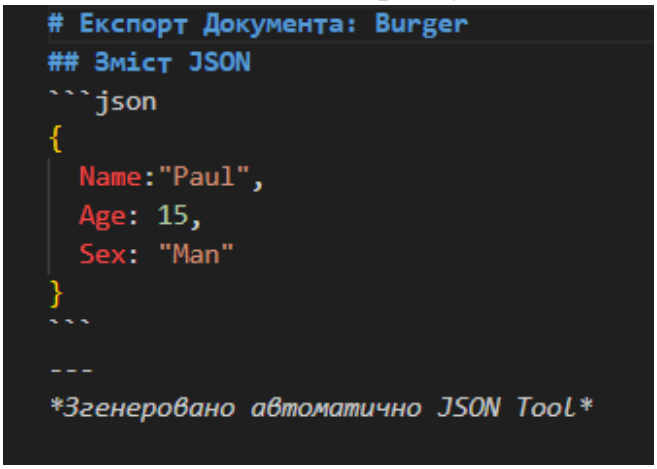
Мої JSON документи

ID	НАЗВА ДОКУМЕНТА	ДАТА СТВОРЕННЯ	ДІЇ
1	Burger	22.10.2025 13:08	[MD] [TXT]
2	test	22.10.2025 13:14	[MD] [TXT]
3	Добрий день	25.10.2025 01:48	[MD] [TXT]
4	Андрій 1	25.10.2025 02:09	[MD] [TXT]
5	Тест	25.10.2025 02:22	[MD] [TXT]
8	Burger	08.11.2025 00:20	[MD] [TXT]

Бачимо, що з’явився рядок «Дії». Є дві кнопки: [MD] та [TXT]. Перша завантажує на наш комп’ютер обраний файл в режимі Markdown, друга в текстовому.



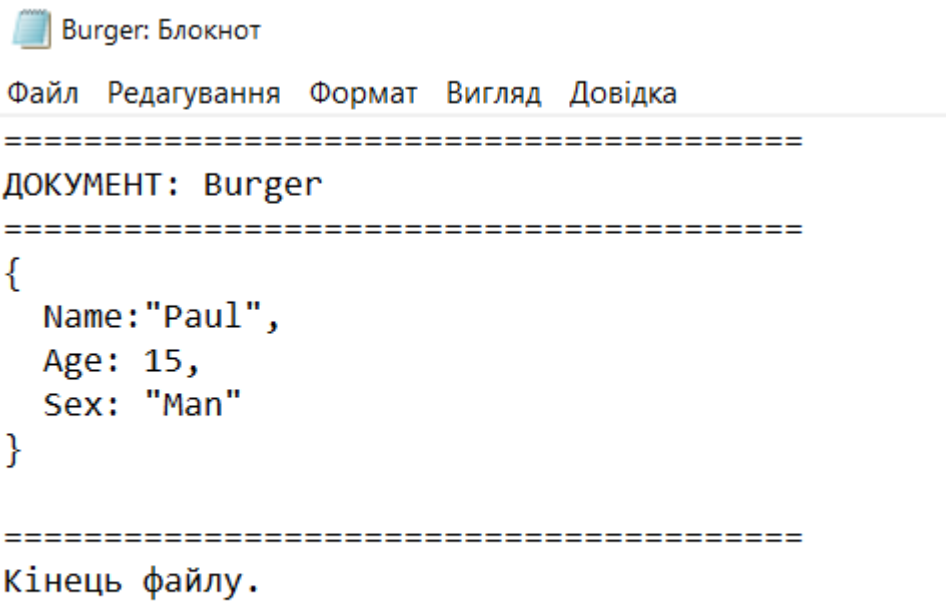
Маємо дані файлу:



```
# Експорт Документа: Burger
## Зміст JSON
```json
{
  Name:"Paul",
  Age: 15,
  Sex: "Man"
}
```

Згенеровано автоматично JSON Tool
```

Формат Markdown



Текстовий формат

5. 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.

Висновок

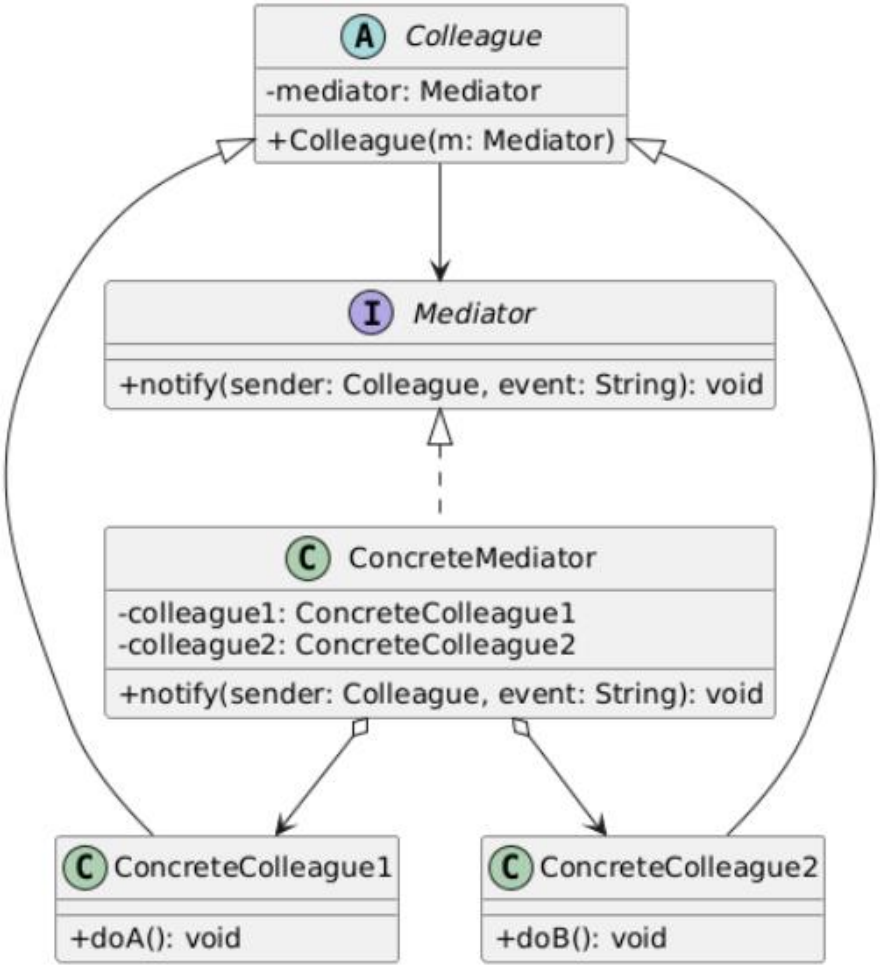
Під час виконання лабораторної роботи я ознайомився з патерном проєктування "Шаблонний метод" (Template Method). Було створено абстрактний клас, який містить шаблонний метод експорту документа. Цей метод визначає загальний алгоритм експорту (заголовок -> тіло -> підвал), але делегує реалізацію конкретних кроків абстрактним методам. Також були створені дві конкретні реалізації: MarkdownExporter — для експорту документа у формат Markdown (вимога теми), та TxtExporter — для експорту у звичайний текстовий файл.

Відповіді на контрольні питання

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» (Mediator) визначає об'єкт, який інкапсулює спосіб взаємодії множини об'єктів. Він забезпечує слабку зв'язаність системи, позбавляючи об'єкти необхідності явно посилатися один на одного і дозволяючи тим самим незалежно змінювати їхню взаємодію.

2. Нарисуйте структуру шаблону «Посередник».



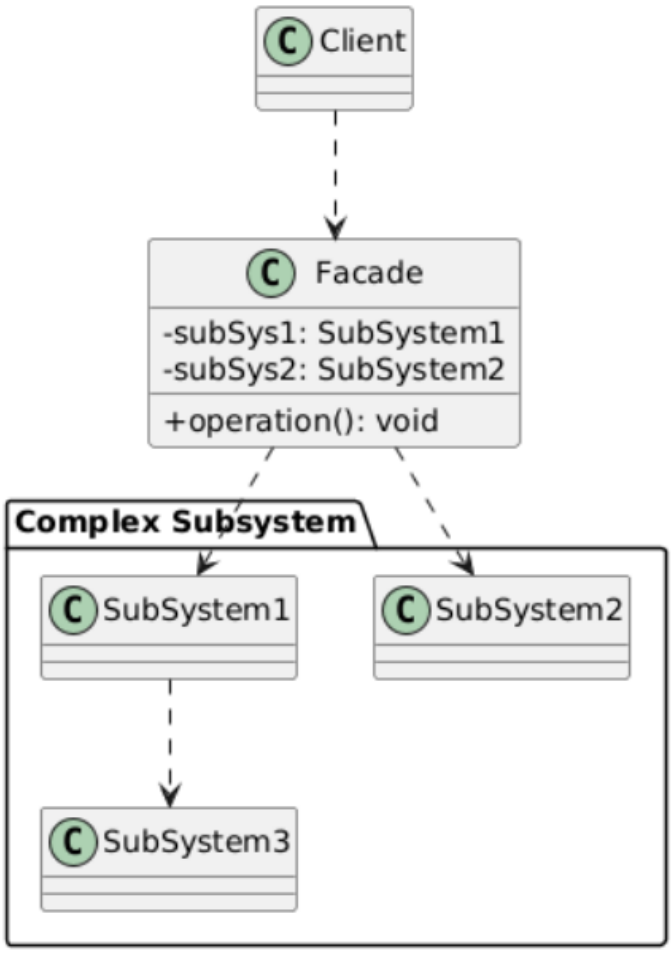
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

1. Mediator (Посередник): Інтерфейс для обміну інформацією між об'єктами-колегами.
2. ConcreteMediator (Конкретний посередник): Реалізує інтерфейс, координує дії кількох об'єктів Colleague. Знає про всіх колег.
3. Colleague (Колега): Абстрактний клас або інтерфейс для компонентів системи. Кожен колега знає лише про свого Посередника.
4. ConcreteColleague: Конкретні компоненти. Вони не спілкуються напряму між собою. Якщо їм щось треба, вони звертаються до Посередника.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» (Facade) надає простий уніфікований інтерфейс до складної підсистеми, що складається з багатьох класів. Він приховує складність системи від клієнта.

5. Нарисуйте структуру шаблону «Фасад».



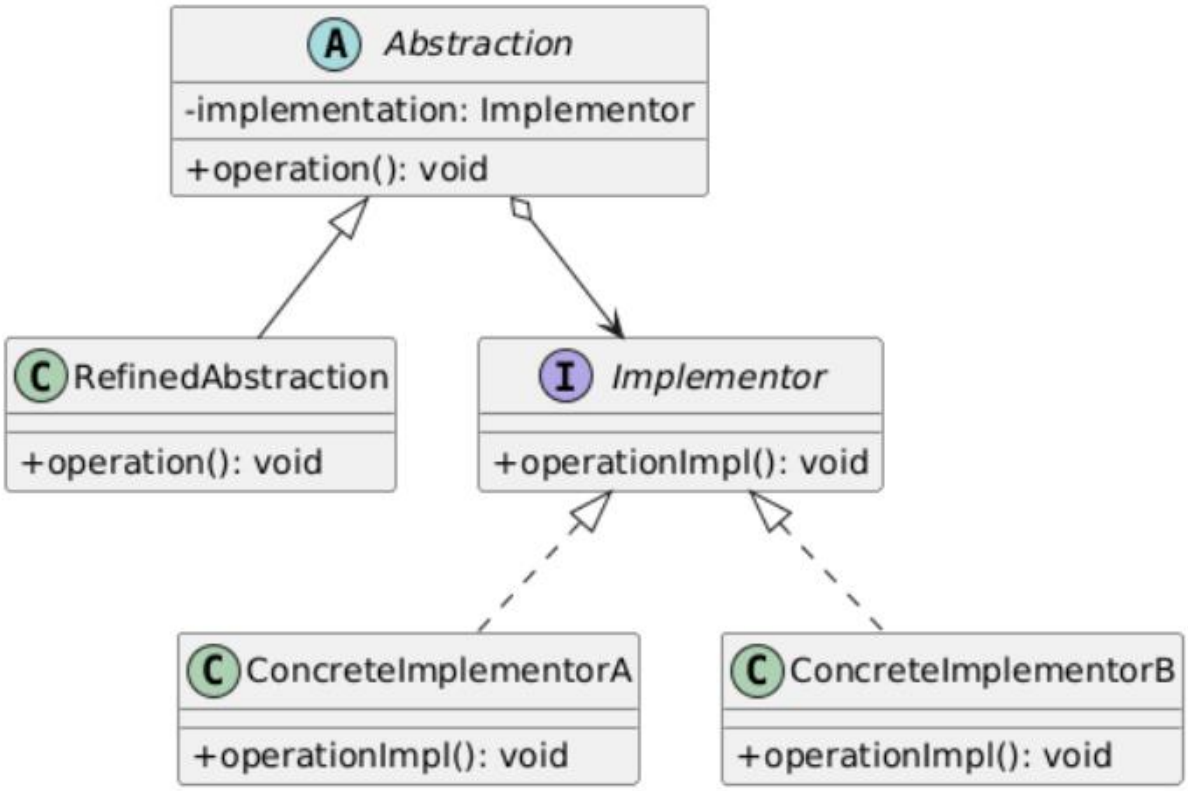
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

1. Facade (Фасад): Клас, який знає, яким класам підсистеми адресувати запит, і делегує їм виконання роботи.
2. Subsystem Classes (Класи підсистеми): Складні класи, які виконують реальну роботу. Вони не знають про існування Фасаду.
3. Client (Клієнт): Працює тільки з Фасадом, не заглиблюючись у деталі підсистеми.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» (Bridge) призначений для відокремлення абстракції від її реалізації, щоб вони могли змінюватися незалежно. Це дозволяє уникнути комбінаторного вибуху класів при розширенні функціоналу у двох вимірах (наприклад, Форма + Колір).

8. Нарисуйте структуру шаблону «Міст».



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

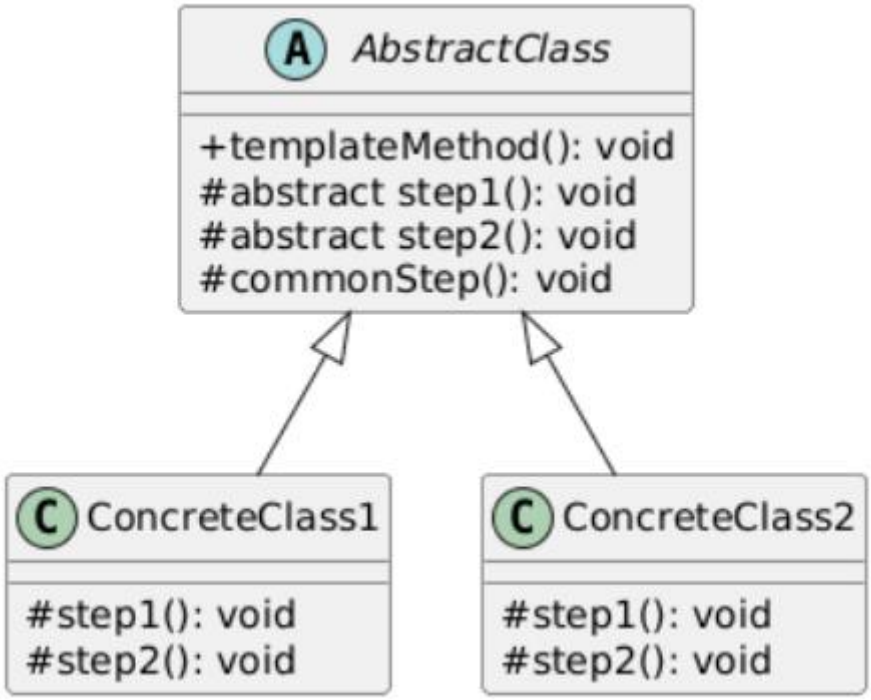
1. Abstraction (Абстракція): Визначає інтерфейс високого рівня і зберігає посилання на об'єкт Implementor.
2. RefinedAbstraction (Уточнена абстракція): Розширює інтерфейс Abstraction.
3. Implementor (Реалізатор): Визначає інтерфейс для класів реалізації.
4. ConcreteImplementor (Конкретний реалізатор): Містить конкретну реалізацію логіки.

Взаємодія: Абстракція делегує роботу об'єкту Реалізатора, який у ній міститься.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблон «Шаблонний метод» (Template Method) визначає скелет алгоритму в базовому класі, але дозволяє підкласам перевизначати певні кроки цього алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблоний метод».



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

1. AbstractClass (Абстрактний клас): Оголошує шаблонний метод (який містить кроки алгоритму) та абстрактні методи (кроки), які мають бути реалізовані.
2. ConcreteClass (Конкретний клас): Успадковує абстрактний клас і реалізує абстрактні методи, надаючи конкретні деталі кроків алгоритму.

Взаємодія: Клієнт викликає шаблонний метод базового класу. Цей метод по черзі викликає кроки алгоритму, частина з яких реалізована в базовому класі, а частина (абстрактна) — делегується конкретному підкласу.

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

- Шаблонний метод — це поведінковий патерн. Він керує алгоритмом (послідовністю дій).
- Фабричний метод — це породжуючий патерн. Він керує створенням об'єктів.
- Часто Фабричний метод може бути одним із кроків (методів) усередині Шаблонного методу.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає функціональність незалежного розширення двох ортогональних ієрархій класів (Абстракції та Реалізації). Це дозволяє змінювати або додавати нові абстракції (наприклад, нові типи вікон) та нові реалізації (наприклад, нові операційні системи), не змінюючи код іншої сторони та не створюючи величезну кількість підкласів.