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

Тема: Патерни проектування.

Виконала
студентка групи ІА–32:
Слюсарева А.А.

Київ 2025

ЗМІСТ

7.1 Завдання

7.2 Теоретичні відомості

7.3 Хід роботи

7.4 Висновок

7.5 Контрольні запитання

ЛАБОРАТОРНА РОБОТА № 7

Тема: Патерни проектування.

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

7.1. 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.

7.2. Теоретичні відомості

Шаблон «Template Method»

Призначення патерну: Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування веб сторінки: 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.

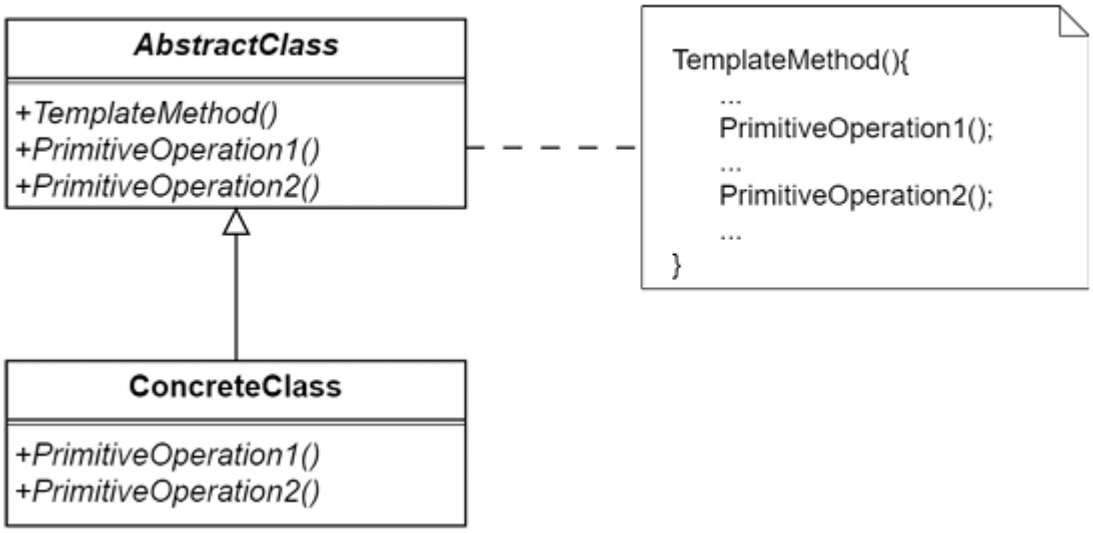


Рисунок 7.4. Структура патерну «Шаблонний метод»

Проблема: Ви працюєте в команді, що займається розробкою застосунку для редагування відео-файлів. Застосунок вже працює з форматом відео MPEG 4, а саме дозволяє читати такі файли, виконувати попередню обробку даних для відображення в відео-редакторі. Ви отримуєте нову задачу на реалізацію можливості роботи з більш старим форматом MPEG-2. Ви бачите два варіанта: зробити копію існуючого класу, що працює з MPEG-4, або вносити зміни в уже існуючий клас. Щоб прийняти рішення ви більш детально розбираєтеся з існуючим алгоритмом і бачите, що близько 70 відсотків коду має бути таким самим. Тому ви вирішуєте змінити вже існуючий клас для роботи з MPEG-4 добавляючи в місцях де це потрібно умови з перевіркою, що якщо формат MPEG-2 то

відпрацьовувати новий код, який ви добавили. Через деякий час, на запити від користувачів, вам на реалізацію приходить задача добавити підтримку ще більш старого формату MPEG-1. Ви вносите зміни так само в існуючий клас, тільки умови стали більш складними, тому що розгалуження логіки йде на три гілки. Ще через деякий час приходить аналогічна задача на додавання читання даних з файлів формату H.262. Ви починаєте працювати над задачею і бачите, що код, який до цього був ще більш-менш зрозумілим стає зовсім важким для читання та внесення змін.

Рішення: Патерн «Шаблонний метод» (Template Method) пропонує загальний алгоритм винести в базовий клас, а частини алгоритма, які для різних задач виконуються по різному, виділити в окремі методи. Ці методи будуть викликатися в алгоритмі, що реалізований в базовому класі. В дочірніх класах ці виділені методи будуть перевизначатися. Таким чином загальна логіка залишається в базовому класі, а специфічна частина реалізується в дочірніх класах. Якщо подивитися на задачу з відео-редактором, то застосування «Шаблонного методу» наведе лад в коді і спростить його зміни. 86 Як це зробити: По перше, в алгоритмі всі блоки коду де є вибір гілки на основі типу формату виділяються в окремі методи. У випадку з відео редактором, це скоріш за все будуть блоки коду пов'язані з читанням даних та розпакування їх в кадри, а також читання звукових доріжок. Далі створюється загальний базовий клас в який переноситься загальний алгоритм, а також об'являються віртуальні методи (фактично беремо сигнатуру тих методів, що виділили на попередньому кроці). Далі створюємо дочірні класи під кожен формат файлу і перевизначаємо віртуальні методи. Фактично при цьому в кожному такому методі в дочірньому класі із реалізації цих методів, що була виділена на першому кроці, залишається код гілки який відповідав вибраному формату. Після всіх цих змін ми маємо реалізацію патерна «Шаблоний метод»: в базовому

класі реалізовано базовий алгоритм (по суті більша частина алгоритму) і в дочірніх класах перевизначені методи зі специфічною логікою. Після таких змін, добавити підтримку нового формату стає легше, тому що достатньо буде добавити лише новий дочірній клас і перевизначити в ньому необхідні методи. Слід зауважити, що якщо у вас алгоритми співпадають більше ніж на 50 відсотків, то застосування шаблонного методу буде доцільним, але якщо у вас алгоритми співпадають лише відсотків на 10 або 20, то скоріш за все, краще буде використати патерн «Стратегія».

Переваги та недоліки:

+ Полегшує повторне використання коду.

- Ви жорстко обмежені скелетом існуючого алгоритму.

- Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.

- З ростом складності загального алгоритму шаблонний метод стає занадто складно підтримувати, особливо, коли є багато віртуальних методів для перевизначення в підкласах.

7.3 Хід роботи

Діаграма класів:

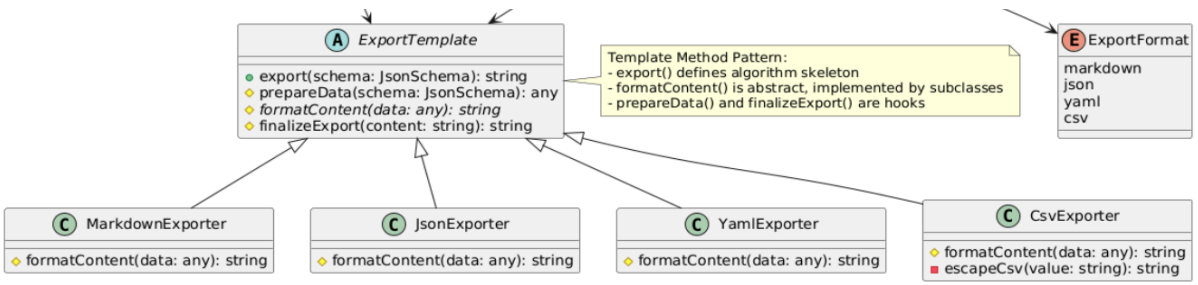


Рис 1. Діаграма класів системи

Фрагменти коду:

```
import { JsonSchema } from
'../../core/models/json-schema.model';

export abstract class ExportTemplate {
 export(schema: JsonSchema): string {
   const prepared = this.prepareData(schema);
   const content = this.formatContent(prepared);
   return this.finalizeExport(content);
 }

 protected prepareData(schema: JsonSchema): any {
   return schema.properties;
 }

 protected abstract formatContent(data: any): string;

 protected finalizeExport(content: string): string {
   return content;
 }
}

import { ExportTemplate } from '../export.template';

export class CsvExporter extends ExportTemplate {
 protected formatContent(data: any): string {
   let csv =
'Name,Description,Type,Format,Example\n';
```

```
    for (const prop of data) {
      const name = this.escapeCsv(prop.name);
      const desc = this.escapeCsv(prop.description ||
'');
      const type = this.escapeCsv(prop.dataType);
      const format = this.escapeCsv(prop.format ||
'');
      const example =
this.escapeCsv(JSON.stringify(prop.example));

      csv +=
`${name},${desc},${type},${format},${example}\n`;
    }

    return csv;
  }

  private escapeCsv(value: string): string {
    if (value.includes(',') || value.includes('"') ||
value.includes('\n')) {
      return `"${value.replace(/"/g, '""')}"`;
    }
    return value;
  }
}

import { ExportTemplate } from '../export.template';
```

```
export class JsonExporter extends ExportTemplate {
 protected formatContent(data: any): string {
   return JSON.stringify(data, null, 2);
 }
}

import { ExportTemplate } from '../export.template';

export class MarkdownExporter extends ExportTemplate
{
 protected formatContent(data: any): string {
   let md = '| Name | Description | Type | Format |
Example |\n';
   md +=
'|------|-------------|------|--------|----------|\n'
;

   for (const prop of data) {
     md += `| ${prop.name} | ${prop.description ||
''} | ${prop.dataType} | ${prop.format || '-'} |
${JSON.stringify(prop.example)} |\n`;
   }

   return md;
 }
}

import { ExportTemplate } from '../export.template';
```

```
export class YamlExporter extends ExportTemplate {
 protected formatContent(data: any): string {
   let yaml = '';

   for (const prop of data) {
     yaml += `- name: ${prop.name}\n`;
     yaml += `  description: ${prop.description ||
''}\n`;
     yaml += `  dataType: ${prop.dataType}\n`;
     if (prop.format) yaml += `  format:
${prop.format}\n`;
     yaml += `  example:
${JSON.stringify(prop.example)}\n`;
   }

   return yaml;
 }
}
```

У реалізації використано патерн Шаблонний метод, який визначає загальну структуру операції експорту в абстрактному класі ExportTemplate. Базовий метод export задає послідовність кроків: підготовка даних, форматування та фінальне оформлення. Конкретні експортери (CsvExporter, JsonExporter, MarkdownExporter, YamlExporter) перевизначають лише крок formatContent, не змінюючи загальний алгоритм. Це дозволяє додавати нові формати експорту без дублювання логіки та без модифікації базового класу.

7.4 Висновок

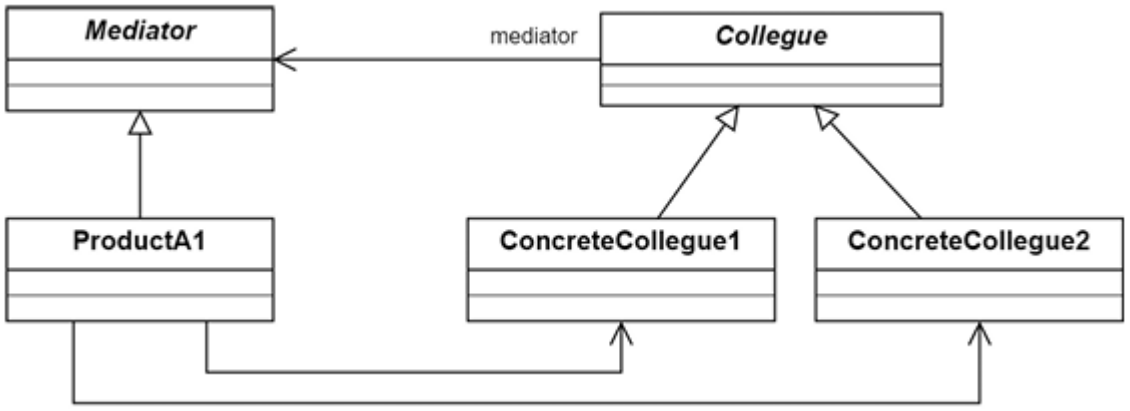
Під час даної роботи було реалізовано патерн Шаблонний метод у додатку JSON Tool.

7.5 Контрольні запитання

1. **Яке призначення шаблону «Посередник»?**

   Посередник зменшує зв’язність між об’єктами, дозволяючи їм взаємодіяти через центральний об’єкт-координатор. Це спрощує систему та робить взаємодію компонентів більш керованою.

2. **Нарисуйте структуру шаблону «Посередник».**



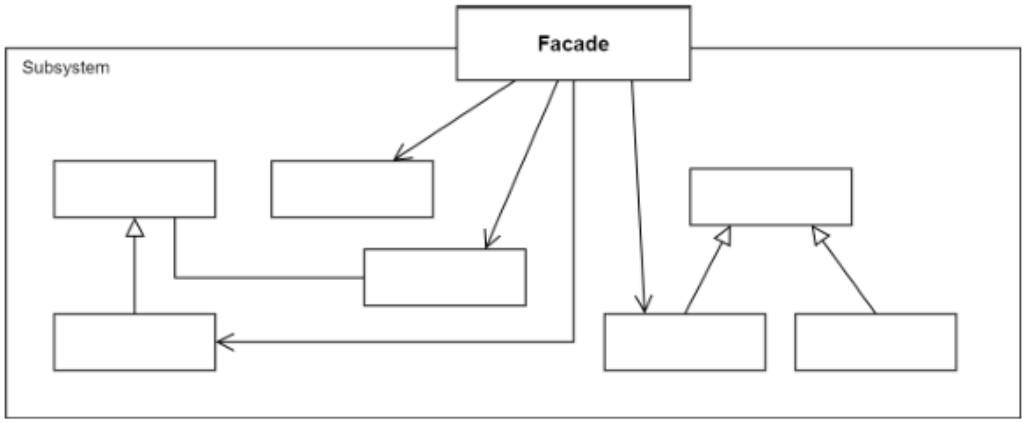
3. **Які класи входять в шаблон «Посередник», та яка між ними взаємодія?**

   У шаблон входять Mediator, ConcreteMediator, Colleague та ConcreteColleague. Об’єкти-співробітники надсилають запити Посереднику, а він вирішує, кому їх передати, керуючи всією взаємодією.

4. **Яке призначення шаблону «Фасад»?**

   Фасад надає спрощений інтерфейс до складної підсистеми, приховуючи її структуру та зменшуючи залежність клієнта від внутрішніх компонентів.

5. **Нарисуйте структуру шаблону «Фасад».**



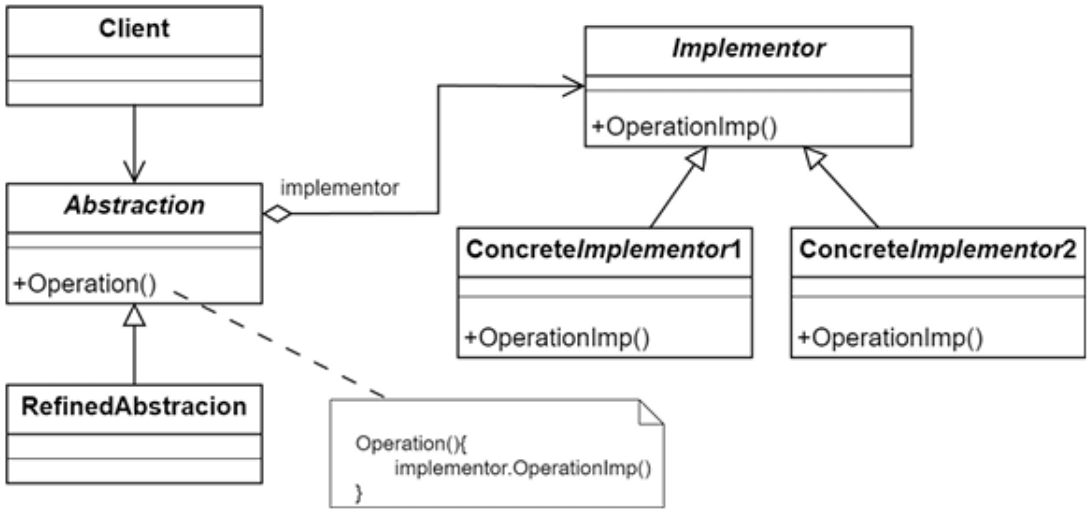
6. **Які класи входять в шаблон «Фасад», та яка між ними взаємодія?**

Фасад взаємодіє з підсистемами, викликаючи їхні методи у потрібній послідовності. Клієнт працює лише з фасадом і не має прямого доступу до підсистем.

7. **Яке призначення шаблону «Міст»?**

Міст відокремлює абстракцію від її реалізації, дозволяючи змінювати їх незалежно одна від одної та створювати різні комбінації.

8. **Нарисуйте структуру шаблону «Міст».**



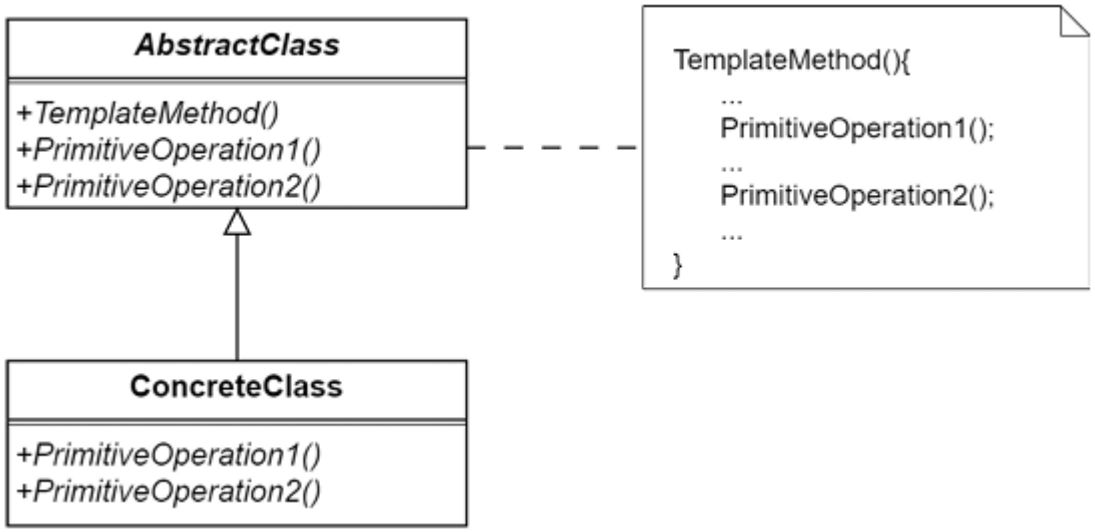
9. **Які класи входять в шаблон «Міст», та яка між ними взаємодія?**

У шаблон входять Abstraction та Implementor з їх конкретними реалізаціями. Абстракція делегує виконання операцій об’єкту реалізації, забезпечуючи гнучкість і незалежність змін.

10. **Яке призначення шаблону «Шаблонний метод»?**

Шаблонний метод визначає загальний алгоритм у базовому класі, дозволяючи підкласам перевизначати окремі кроки алгоритму без зміни його структури.

11. **Нарисуйте структуру шаблону «Шаблонний метод».**



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

AbstractClass містить шаблонний метод, що визначає алгоритм. ConcreteClass перевизначає окремі кроки алгоритму, зберігаючи його загальну послідовність.

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

Шаблонний метод визначає структуру алгоритму та дозволяє перевизначати його кроки, тоді як фабричний метод визначає інтерфейс для створення об’єктів і дозволяє підкласам вирішувати, який саме об’єкт створювати.

14. **Яку функціональність додає шаблон «Міст»?**

Міст додає можливість незалежно розширювати ієрархії абстракцій і реалізацій, комбінуючи їх без створення вибухової кількості підкласів.