Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №7

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проєктування»

Варіант: E-mail клієнт

Виконав:
студент групи ІА-32
Карповець Роман

Київ - 2025

Тема: Патерни проєктування.

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

Тема проєкту: E-mail клієнт

Опис: Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Зміст

Теоретичні відомості

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.

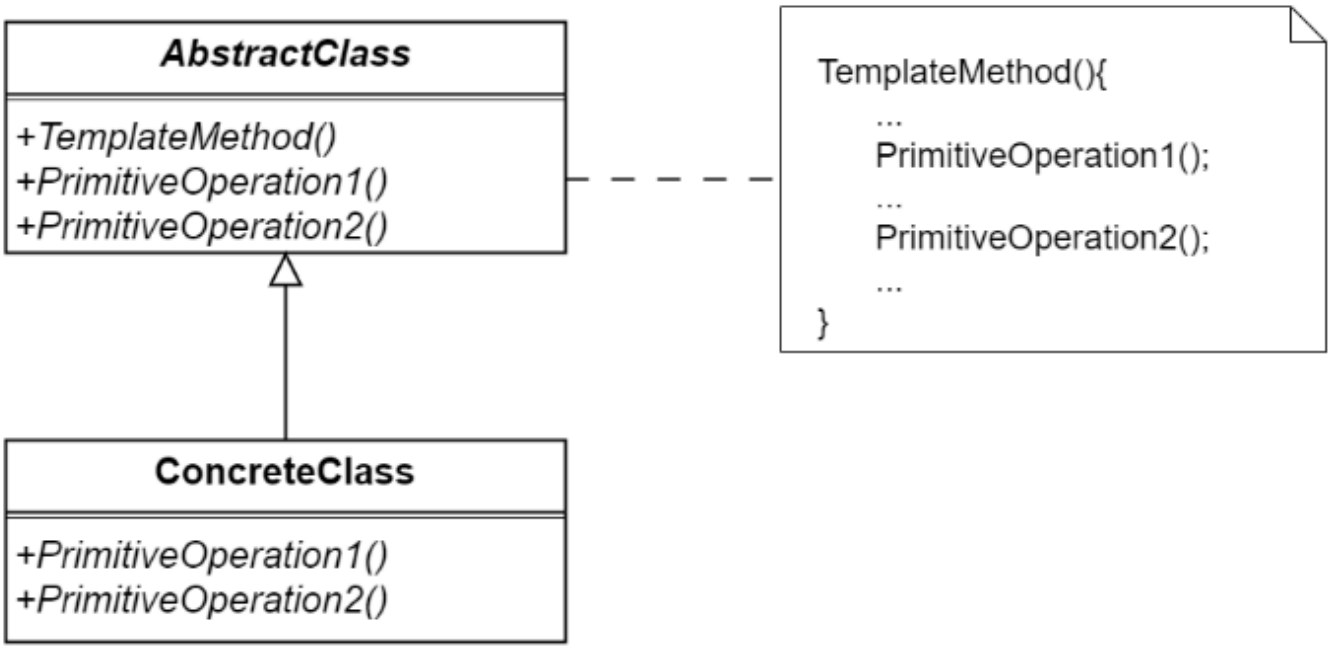

Рис. 1. Структура патерну «Шаблонний метод»

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.

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.

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.

Хід роботи

Реалізація патерну Template Method

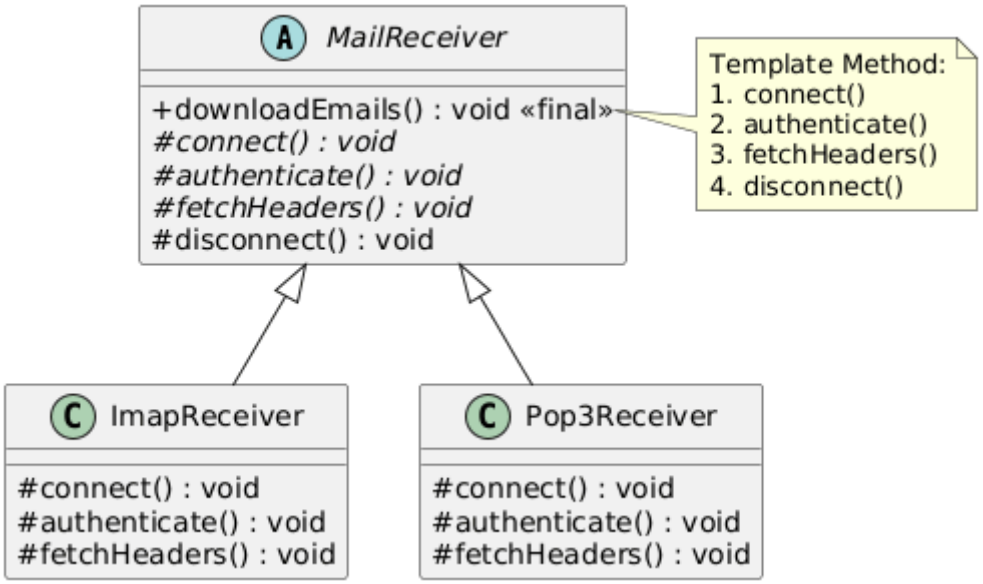

Рис. 2. Діаграма класів з використанням патерну Template Method

Фрагмент коду з реалізацією патерну TemplateMethod

MailReceiver.java

```java
package com.emailclient.service.receiver;

/**
* Pattern: Template Method
*/
public abstract class MailReceiver {

    // Шаблонний метод - final, щоб алгоритм не можна було змінити
    public final void downloadEmails() {
        System.out.println("--- Start downloading via " + getProtocolName() + " ---");
        connect();
        authenticate();
        fetchHeaders();
        disconnect();
        System.out.println("--- Finished ---\n");
    }

    // Абстрактні методи (кроки), які мають реалізувати підкласи
    protected abstract void connect();
    protected abstract void authenticate();
    protected abstract void fetchHeaders();
```

```
    protected abstract String getProtocolName();

    // Загальний метод (hook), однаковий для всіх (можна перевизначити за бажанням)
    protected void disconnect() {
        System.out.println("System: Closing connection securely.");
    }
}
```

ImapReceiver.java

```
package com.emailclient.service.receiver;

public class ImapReceiver extends MailReceiver {

    @Override
    protected String getProtocolName() {
        return "IMAP";
    }

    @Override
    protected void connect() {
        System.out.println("IMAP: Connecting to port 993 (SSL)...");
    }

    @Override
    protected void authenticate() {
        System.out.println("IMAP: Sending OAuth2 token...");
    }

    @Override
    protected void fetchHeaders() {
        System.out.println("IMAP: Syncing folder structure (Inbox, Sent, Trash)...");
        System.out.println("IMAP: Fetching only message headers (lazy loading).");
    }
}
```

Pop3Receiver.java

```
package com.emailclient.service.receiver;

public class Pop3Receiver extends MailReceiver {

    @Override
    protected String getProtocolName() {
```

```
        return "POP3";
    }

    @Override
    protected void connect() {
        System.out.println("POP3: Connecting to port 995...");
    }

    @Override
    protected void authenticate() {
        System.out.println("POP3: Authenticating with USER/PASS command...");
    }

    @Override
    protected void fetchHeaders() {
        System.out.println("POP3: Downloading ALL messages to local storage...");
        System.out.println("POP3: Marking messages for deletion on server.");
    }
}
```

Висновок: я вивчив структуру шаблонів «Mediator», «Facade», «Bridge», «Template method» та навчився застосовувати їх в реалізації програмної системи.