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 «Теорія розробки програмного забезпечення»

Тема: «Патерни проектування»

Виконала:

студентка групи ІА-33

Піголь Тетяна

Перевірив:

Асистент кафедри ІСТ

Мягкий Михайло Юрійович

Київ – 2025

Тема: Патерни проектування.

Тема проєкту:

E-mail клієнт (singleton, builder, decorator, template method, interpreter, client-server)

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.

1. **Короткі теоретичні відомості:**

Шаблон «Mediator»

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Це поведінковий патерн проєктування, який зменшує зв'язність (coupling) між безліччю класів, виносячи їхню взаємодію в один центральний об'єкт-посередник. Замість того щоб об'єкти спілкувалися один з одним напряму (що створює заплутану "павутину"), вони спілкуються лише з Посередником, а він вже вирішує, кому і що передати.

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».

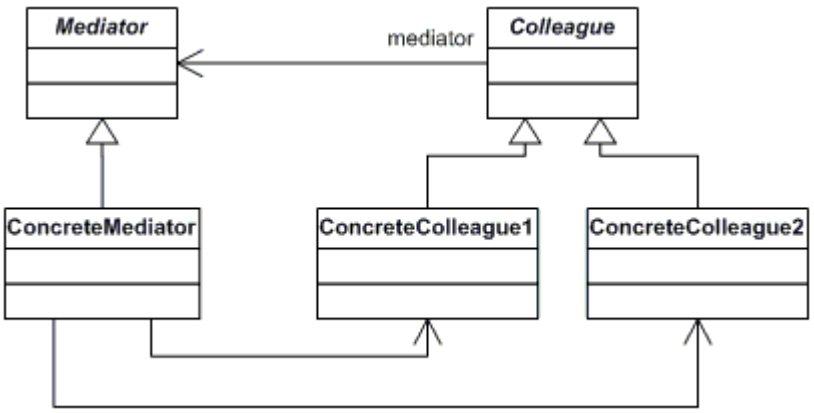

Рис. 1.1 – Структура патерну «Медіатор»

Структура патерну

1. Mediator (Інтерфейс): Описує методи для комунікації з компонентами (зазвичай метод типу notify(sender, event)).
2. ConcreteMediator (Конкретний посередник): Реалізує логіку взаємодії. Він знає про всі конкретні компоненти та керує ними.
3. Colleagues (Колеги/Компоненти): Це класи, які мають посилання на Медіатора. Вони не знають про інші компоненти.

Переваги та недоліки:

+ Організація взаємодії між об’єктами лише через посередника спрощує розуміння та супроводження такого коду.

+ Додавання нових посередників без зміни існуючих компонентів дозволяє розширювати систему без зміни існуючого коду.

+ Зменшення залежностей між об’єктами підвищує гнучкість системи.

- Посередник, з часом, може перетворитися на дуже складний об’єкт, який робить все («God Object»).

Шаблон «Facade»

Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагеті-коду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей. Це структурний патерн проектування, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку.

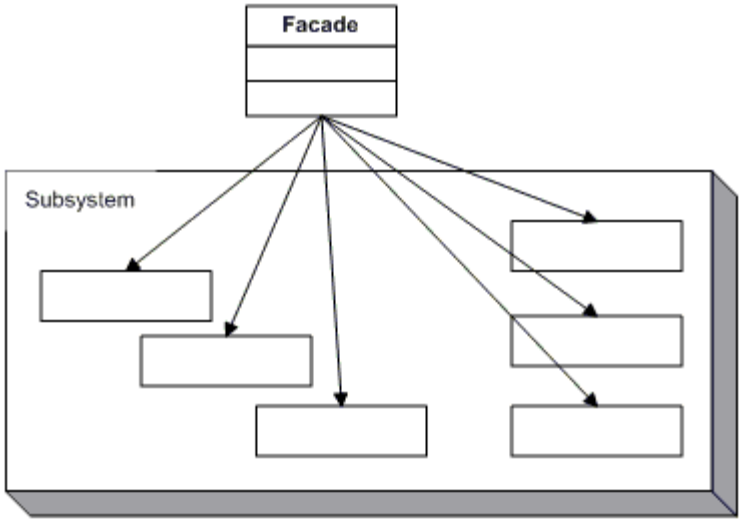

Рис. 1.2 – Структура патерну «Facade»

У розробці часто буває ситуація, коли коду потрібно працювати з великою кількістю об'єктів складної бібліотеки. У такому випадку треба їх ініціалізувати, слідкувати за порядком виконання, залежностями тощо. Це призводить до того, що бізнес-логіка класу тісно переплітається з деталями реалізації сторонньої бібліотеки, код стає важко читати та підтримувати. При такі проблемі, створюється один клас (Фасад), який має спрощені методи. Всередині цього методу Фасад сам знає, які класи підсистеми викликати і в якому порядку.

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.

Шаблон «Bridge»

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. Міст – це структурний патерн проектування, який розділяє один великий клас або набір тісно пов'язаних класів на дві окремі ієрархії:

абстракцію та реалізацію, дозволяючи змінювати їх незалежно одна від одної. Коротко «Розділяй та володарюй». Замість спадкування використовується композиція.

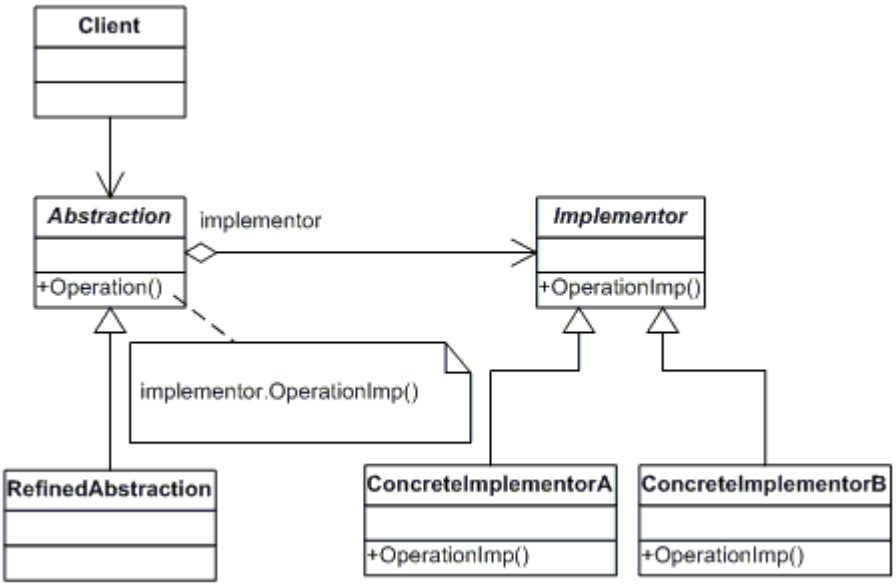

Рис. 1.3 – Структура патерну «Bridge»

Структура патерна:

1. Abstraction (Абстракція): Високорівневий шар логіки. Вона керує, але сама роботу не виконує.
2. Implementation (Реалізація): Інтерфейс для низькорівневих частин.
3. Refined Abstraction (Уточнена абстракція): Підкласи абстракції.
4. Concrete Implementation (Конкретна реалізація): Реальний код, що робить роботу.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Шаблон «Template Method»

Шаблонний метод (Template Method) – це поведінковий патерн проектування, який визначає «скелет» алгоритму в суперкласі, але дозволяє підкласам перевизначати певні кроки цього алгоритму, не змінюючи його загальної структури. Шаблон «Template Method» дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам.

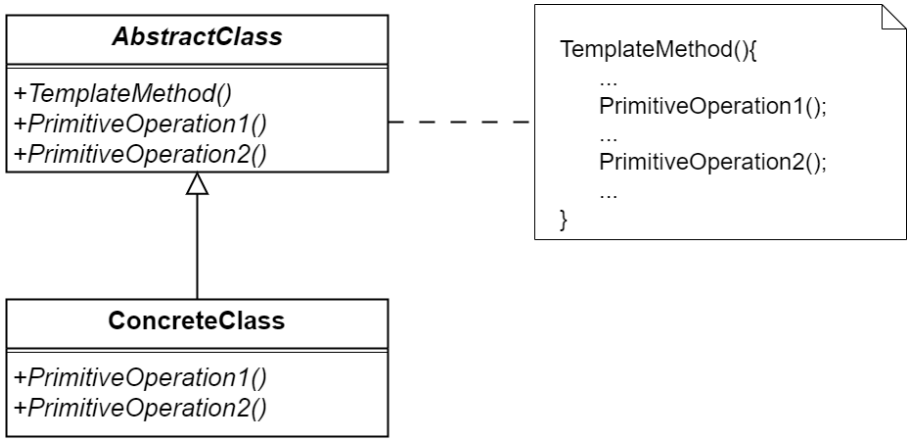

Рис. 1.4 – Структура патерну «Template Method»

Структура патерна:

1. Шаблонний метод: Головний метод, який викликає інші методи в певному порядку. Часто його роблять final, щоб підкласи не могли зламати структуру алгоритму.
2. Абстрактні методи: Кроки, які обов'язково мають реалізувати підкласи.
3. Конкретні методи: Загальні кроки, однакові для всіх.
4. Хуки (Hooks): Методи з порожньою реалізацією за замовчуванням. Підкласи можуть їх перевизначити, якщо хочуть, але не зобов'язані.

Слід зауважити, що якщо 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.

2. Хід роботи:

Для реалізації процес відправки електронного листа можна використати патерн Template Method (Шаблонний метод). Було створено структуру класів, яка стандартизує цей процес, але дозволить кожному поштовому провайдеру (Gmail, Ukr.net, i.ua) надавати свої специфічні налаштування (хост, порт, налаштування SSL).

Цей патерн ідеально підходить для даної задачі, оскільки алгоритм відправки (з'єднання -> авторизація -> відправка) однаковий, а параметри різні.

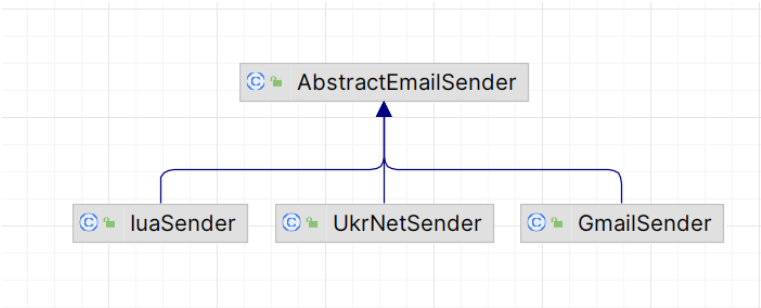

Рис. 2.1 – Діаграма класів реалізації патерну Template Method.

На рис. 2.1 зображено абстрактний клас AbstractEmailSender, який визначає загальний алгоритм відправки листів, та конкретні класи-реалізації (IuaSender, UkrNetSender, GmailSender), що містять специфічні налаштування для відповідних поштових провайдерів.

```java
public abstract class AbstractEmailSender {

    1 usage
    public final void sendEmail(Mail mail, String password) {
        System.out.println("=== Початок відправки листа через " + getProviderName() + " ===");

        validate(mail);

        Properties props = new Properties();
        props.put("mail.smtp.host", getSmtpHost());
        props.put("mail.smtp.port", String.valueOf(getSmtpPort()));
        props.put("mail.smtp.ssl.enable", String.valueOf(isSslEnabled()));
        props.put("mail.smtp.auth", "true");

        System.out.println("Налаштування: " + props);

        connect(mail.getSender(), password);

        transport(mail);

        disconnect();

        System.out.println("=== Лист успішно відправлено ===");
    }
```

Рис. 2.2 – Частина реалізація класу AbstractEmailSender.

AbstractEmailSender – це абстрактний батьківський клас, який містить шаблонний метод sendEmail(). Цей метод визначає скелет алгоритму відправки листа (валідація, отримання налаштувань, з'єднання, передача даних, роз'єднання).

```java
@Component
public class GmailSender extends AbstractEmailSender {
    2 usages
    @Override
    protected String getSmtpHost() { return "smtp.gmail.com"; }

    1 usage
    @Override
    protected int getSmtpPort() { return 587; }

    1 usage
    @Override
    protected boolean isSslEnabled() { return true; }

    1 usage
    @Override
    protected String getProviderName() { return "Gmail"; }
}
```

Рис. 2.3 – Реалізація конкретного класу GmailSender.

Конкретні класи (IuaSender, UkrNetSender, GmailSender) наслідують абстрактний клас та реалізують лише змінні кроки алгоритму. Зокрема, вони перевизначають абстрактні методи getSmtpHost() та getSmtpPort(), надаючи унікальні параметри підключення для кожного сервісу.

Така структура дозволяє уникнути дублювання коду, оскільки вся логіка відправки буде зосереджена в одному місці, а додавання нового провайдера потребує лише створення одного нового класу.

Посилання на GitHub: https://github.com/iatan33/EmailApp7

3. Контрольні запитання:

1. Яке призначення шаблону «Посередник»?

Призначення шаблону «Посередник» зменшити хаотичні зв'язки між об'єктами. Він інкапсулює спосіб взаємодії безлічі об'єктів, змушуючи їх спілкуватися не напряму один з одним, а через спеціальний об'єкт-посередник. Це дозволяє змінювати схему взаємодії, не чіпаючи самі об'єкти.

2. Нарисуйте структуру шаблону «Посередник».

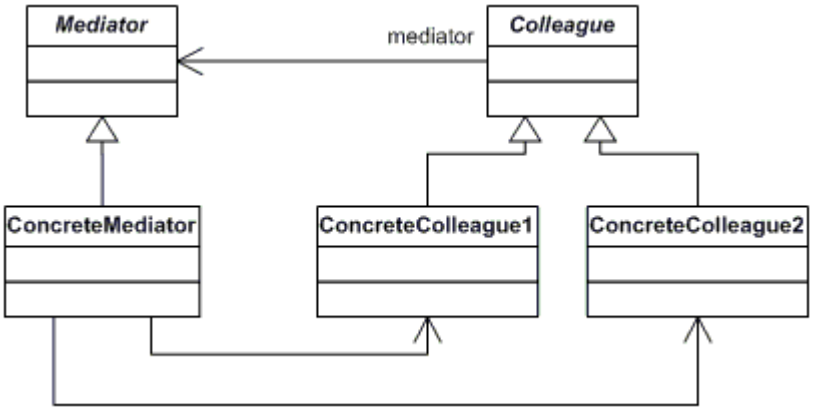

Рис. 3.1 – Структура патерну «Посередник»

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

Mediator (Інтерфейс Посередника): Описує методи для спілкування з компонентами (зазвичай метод notify(sender, event)).

ConcreteMediator (Конкретний Посередник): Реалізує логіку координації. Він знає про всі конкретні компоненти та керує потоком даних між ними.

Colleague / Component (Компонент): Базовий клас або інтерфейс для компонентів. Кожен компонент має посилання на Посередника.

ConcreteColleague (Конкретний Компонент): Реальні об'єкти бізнес-логіки.

Компоненти не знають один про одного. Коли щось трапляється, Компонент повідомляє Посередника. Посередник вирішує, кому передати цю інформацію та які інші Компоненти активувати.

4. Яке призначення шаблону «Фасад»?

Надати простий, уніфікований інтерфейс до складного набору інтерфейсів у підсистемі. Фасад визначає інтерфейс вищого рівня, який полегшує використання підсистеми, приховуючи її складність.

5. Нарисуйте структуру шаблону «Фасад».

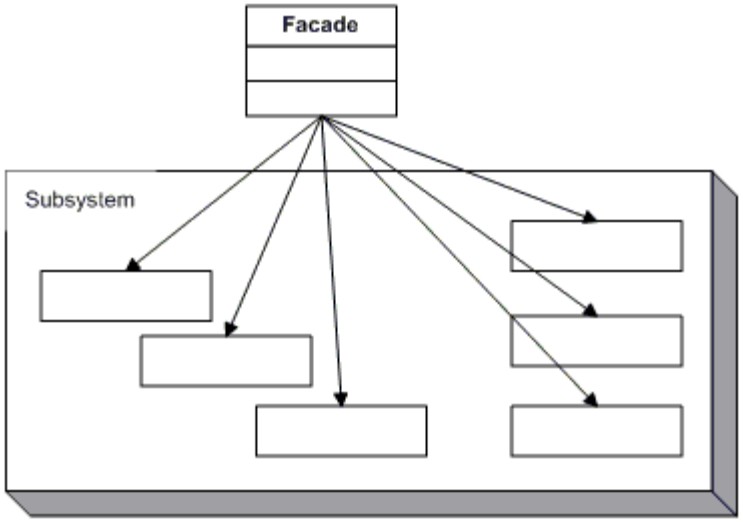

Рис. 3.2 – Структура патерну «Фасад»

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

Facade (Фасад): Клас, який знає, яким класам підсистеми потрібно адресувати запит, і делегує їм роботу.

Subsystem Classes (Класи підсистеми): Набір складних класів (бібліотека, фреймворк), які виконують реальну роботу. Вони не знають про існування Фасаду.

Клієнт звертається до методів Фасаду. Фасад перетворює цей запит на серію викликів до об'єктів підсистеми. Клієнт ізольований від складності підсистеми.

7. Яке призначення шаблону «Міст»?

Призначення шаблону «Міст» – відокремити абстракцію від її реалізації так, щоб вони могли змінюватися незалежно одна від одної. Це дозволяє уникнути

комбінаторного вибуху кількості класів при розширенні системи в двох незалежних вимірах.

8. Нарисуйте структуру шаблону «Міст».

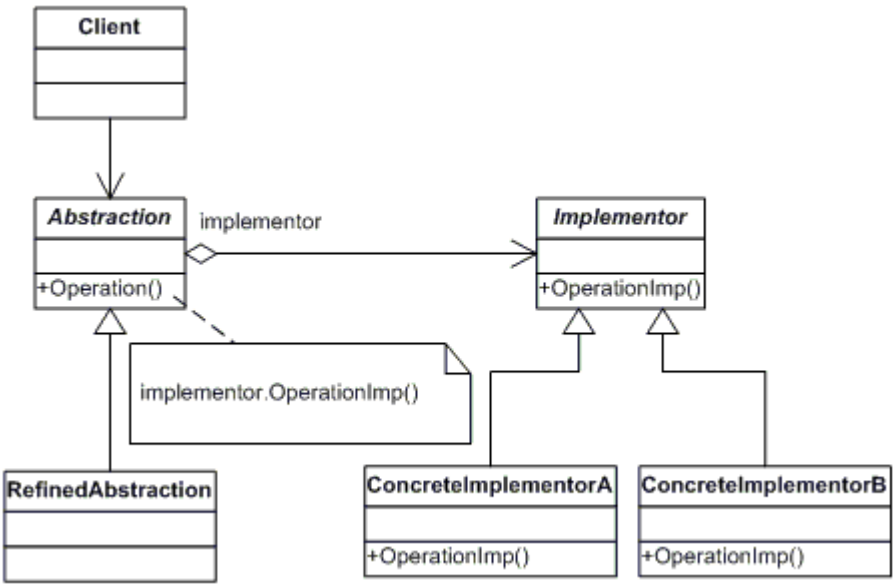

Рис. 3.3 – Структура патерну «Міст»

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

Abstraction (Абстракція): Визначає інтерфейс високого рівня (керуюча логіка). Зберігає посилання на об'єкт Implementor.

RefinedAbstraction (Уточнена абстракція): Розширює інтерфейс Абстракції.

Implementor (Реалізатор): Інтерфейс для низькорівневих операцій (примітиви).

ConcreteImplementor (Конкретний реалізатор): Реальна реалізація.

Абстракція не виконує роботу сама, а делегує її об'єкту Implementor, який міститься в ній (композиція/агрегація).

10. Яке призначення шаблону «Шаблоний метод»?

Патерн «Шаблоний метод» використовується щоб визначити скелет алгоритму в методі суперкласу, перекладаючи реалізацію деяких кроків на підкласи. Шаблон дозволяє підкласам перевизначити певні етапи алгоритму без зміни його загальної структури.

11. Нарисуйте структуру шаблону «Шаблоний метод».

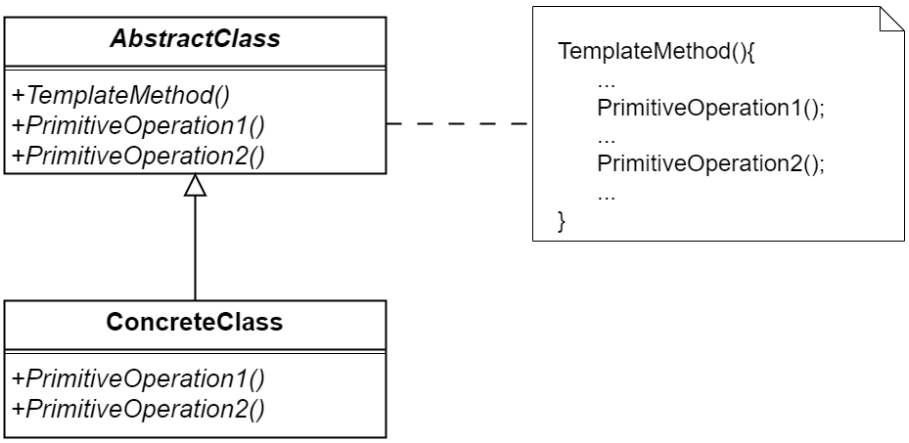

Рис. 3.4 – Структура патерну «Шаблоний метод»

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

AbstractClass (Абстрактний клас): Оголошує шаблонний метод (який містить виклики кроків) та абстрактні методи для кроків, які мають бути реалізовані.

ConcreteClass (Конкретний клас): Реалізує специфічні кроки алгоритму.

Клієнт викликає шаблонний метод батьківського класу. Цей метод виконує кроки алгоритму в заданому порядку, викликаючи реалізації методів у дочірньому класі.

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

Відмінність даних шаблонів полягає в призначенні, вони мають різні цілі на меті. Шаблонний метод — це поведінковий патерн, який керує потоком виконання всього алгоритму (послідовністю дій), а Фабричний метод це твірний (creational) патерн, який відповідає лише за один крок: створення об'єкта. Часто Фабричний метод виступає одним із кроків всередині Шаблонного методу. Наприклад, у шаблоні "Оформлення замовлення" крок "Створити об'єкт доставки" може бути реалізований через Фабричний метод.

Шаблонний метод працює на рівні методу (змінює частини алгоритму), тоді як Фабричний метод часто використовується для того, щоб класи могли делегувати інстанціювання об'єктів підкласам.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» забезпечує динамічну зміну реалізації. Завдяки цьому патерну можна змінити реалізацію (наприклад, перемкнути драйвер або тему оформлення) прямо під час виконання програми, просто замінивши об'єкт Implementor в Абстракції. Також є можливість розширювати класи функціоналу (Абстракції) та класи платформи (Реалізації) незалежно, не ламаючи існуючий код.

Висновки: у ході виконання даної лабораторної роботи було продовжено розробку поштового клієнта та реалізовано програмний код системи на основі діаграм, створених у попередніх роботах. Було успішно реалізовано патерн проєктування Template Method (Шаблонний метод) для реалізації механізму відправки повідомлень через різних поштових провайдерів. Створено абстрактний клас, який визначає загальний алгоритм відправки, та конкретні класи-реалізації, що містять унікальні налаштування для кожного сервісу. Це дозволило уникнути дублювання коду та забезпечити легку розширюваність системи при додаванні нових провайдерів.