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 розробки програмного забезпечення

«Патерни проєктування»

«Веб-браузер»

Виконав:
студент групи ІА-32
Самойленко С. Д.

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Патерни проєктування

Мета: Вивчити структуру шаблонів “Mediator”, “Facade”, “Bridge”, “Template method” та навчитися застосовувати їх в реалізації програмної частини.

Зміст

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

Mediator (Посередник)

Цей патерн поведінки призначений для зменшення хаотичних зв'язків між об'єктами. Замість того, щоб об'єкти (колеги) спілкувалися один з одним напряму, створюючи складну мережу залежностей, вони взаємодіють лише через спеціальний об'єкт-посередник. Посередник інкапсулює логіку взаємодії та

вирішує, якому компоненту передати запит або як на нього відреагувати. Це дозволяє послабити зв'язність коду, оскільки компоненти нічого не знають про реалізацію інших компонентів, а зміна логіки спілкування відбувається лише в одному класі посередника, не зачіпаючи решту системи.

Facade (Фасад)

Фасад — це структурний патерн, який надає простий інтерфейс до складної системи класів, бібліотеки або фреймворку. Він приховує всю складність ініціалізації, налаштування та взаємодії внутрішніх об'єктів підсистеми за одним класом-обгорткою. Клієнтський код взаємодіє тільки з фасадом, викликаючи прості методи, а фасад вже перенаправляє ці виклики потрібним об'єктам у правильному порядку. Це дозволяє відокремити бізнес-логіку програми від деталей реалізації сторонніх бібліотек та спростити використання системи для клієнта, хоча іноді це може обмежувати гнучкість, якщо потрібен доступ до низькорівневих функцій.

Bridge (Міст)

Цей структурний патерн використовується для розділення абстракції та її реалізації так, щоб вони могли змінюватися незалежно одна від одної. Це особливо корисно, коли клас має кілька вимірів розширення (наприклад, "Форма" і "Колір"), що при звичайному успадкуванні призвело б до створення величезної кількості підкласів для кожної комбінації. Міст пропонує замінити успадкування композицією: виділити одну з ієрархій в окремий набір класів (реалізацію) і посилатися на об'єкт цього набору з основної ієрархії (абстракції). Таким чином, ви можете змінювати або додавати нові класи в обидві ієрархії, не ламаючи існуючий код.

Template Method (Шаблонний метод)

Це патерн поведінки, який визначає скелет алгоритму в суперкласі, але дозволяє підкласам перевизначати певні кроки цього алгоритму без зміни його

структури. У базовому класі створюється метод, який викликає ряд інших методів у чітко визначеній послідовності. Частина цих методів може мати реалізацію за замовчуванням, а частина може бути абстрактною. Підкласи реалізують або перевизначають ці конкретні кроки, наповнюючи шаблон своєю логікою, але загальний порядок виконання операцій залишається незмінним, що сприяє повторному використанню коду.

Тема проєкту

6. Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p) Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) – переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Патерн: Template method

Посилання на Github: https://github.com/brxhh/TRPZ

Хід роботи

Діаграма класів

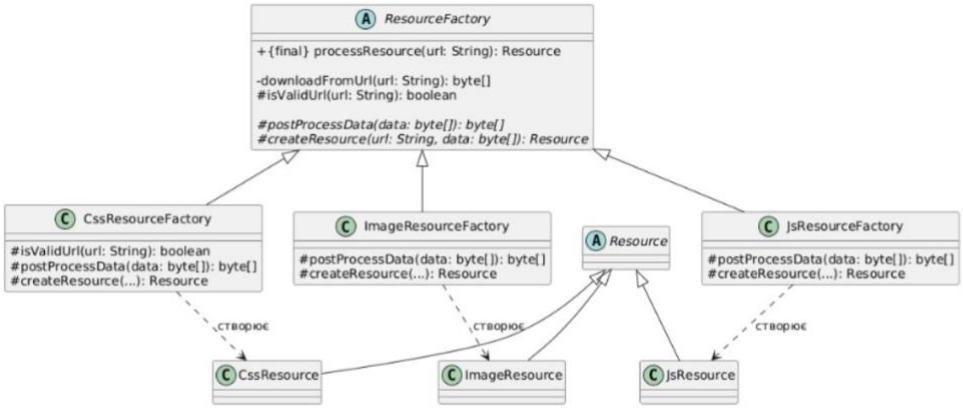


Рис.1 – Діаграма класів для ResourceFactory (патерн Template method)

Опис діаграми класів

Ця діаграма класів ілюструє, як загальний алгоритм обробки ресурсів інкапсульовано в базовому класі за допомогою патерну "Шаблонний метод" (Template Method).

Замість того, щоб кожен конкретний клас фабрики (CssResourceFactory, ImageResourceFactory тощо) повністю дублював логіку завантаження та обробки, спільний "скелет" алгоритму винесено в абстрактний батьківський клас, а специфічні кроки (як-от пост-обробка даних) делегуються підкласам.

Учасники патерну "Шаблонний метод"

На діаграмі чітко видно всіх ключових учасників патерну:

1. Абстрактний Клас (Abstract Class)

- Клас: ResourceFactory
- Роль: Це центральний елемент патерну. Він визначає структуру алгоритму та містить як реалізовані методи, так і абстрактні, які мають бути реалізовані нащадками.
- Ключова деталь:
    - processResource(): Це власне Шаблонний метод. Він позначений як {final}, щоб підкласи не могли змінити послідовність виконання кроків.
    - downloadFromUrl(): Приватний метод із спільною логікою для всіх.
    - isValidUrl(): Метод-перехоплювач (Hook), який має стандартну реалізацію, але може бути перевизначений.

2. Конкретні Класи (Concrete Classes)

- Класи: CssResourceFactory, JsResourceFactory, ImageResourceFactory
- Роль: Це класи, які реалізують або перевизначають специфічні кроки алгоритму, не змінюючи його загальної структури.
- Взаємодія:
    - Вони реалізують абстрактний метод postProcessData(), додаючи унікальну логіку обробки для кожного типу (наприклад, мініфікація для CSS або перевірка безпеки для JS).
    - Вони реалізують метод createResource() (який є Фабричним методом) для створення відповідного об'єкта продукту.

3. Шаблонний метод (The Template Method)

- Метод: processResource(url: String)
- Роль: Визначає скелет алгоритму. Він викликає методи-кроки у фіксованому порядку:
    1. Перевірка валідності (isValidUrl).
    2. Завантаження даних (downloadFromUrl).
    3. Обробка даних (postProcessData).
    4. Створення об'єкта (createResource).

4. Кроки алгоритму (Steps)

- Абстрактні методи: postProcessData, createResource. Це кроки, реалізація яких відкладена до підкласів.
- Конкретні методи: downloadFromUrl. Це кроки, спільні для всіх.
- Хуки (Hooks): isValidUrl. Це кроки, які мають дефолтну поведінку, але можуть бути змінені.

Як працює потік (The Flow)

1. Клієнт викликає метод processResource("style.css") у екземпляра CssResourceFactory.
2. Виконується код методу processResource у батьківському класі ResourceFactory.
3. Спочатку викликається isValidUrl(). Якщо це CSS-фабрика, вона може мати власну перевірку розширення файлу.
4. Потім викликається спільний метод downloadFromUrl(), який завантажує байти.
5. Далі викликається абстрактний метод postProcessData(). Оскільки ми в контексті CssResourceFactory, виконується саме її версія цього методу (наприклад, мініфікація тексту).
6. Нарешті, викликається createResource(), який створює та повертає готовий

об'єкт CssResource.

Частина коду програми з використанням патерну Template Method

ResourceFactory.java

```
package com.edu.web.restservicewebbrowser.factory.resource;

import com.edu.web.restservicewebbrowser.domain.resource.Resource;

public abstract class ResourceFactory {

    public final Resource processResource(String url) {
        System.out.println("\n[Template Method] Початок обробки: " + url);

        if (!isValidUrl(url)) {
            throw new IllegalArgumentException("Невалідний URL або тип файлу: " +
url);
        }

        byte[] rawData = downloadFromUrl(url);

        byte[] processedData = postProcessData(rawData);

        Resource resource = createResource(url, processedData);

        System.out.println("[Template Method] Ресурс успішно створено.");
        return resource;
    }

    protected boolean isValidUrl(String url) {
        return true;
    }

    private byte[] downloadFromUrl(String url) {
        System.out.println(" -> Завантаження байтів з мережі...");
        return ("Content of " + url).getBytes();
    }
```

```
    protected abstract byte[] postProcessData(byte[] data);

    protected abstract Resource createResource(String url, byte[] data);
}
```

JsResourceFactory.java

```
package com.edu.web.restservicewebbrowser.factory.resource;

import com.edu.web.restservicewebbrowser.domain.resource.JsResource;
import com.edu.web.restservicewebbrowser.domain.resource.Resource;

public class JsResourceFactory extends ResourceFactory {

    @Override
    protected byte[] postProcessData(byte[] data) {
        System.out.println(" -> [JS] Перевірка скрипта на віруси...");
        return data;
    }

    @Override
    protected Resource createResource(String url, byte[] rawData) {
        return new JsResource(url, new String(rawData));
    }
}
```

ImageResourceFactory.java

```
package com.edu.web.restservicewebbrowser.domain.resource;

import com.edu.web.restservicewebbrowser.visitor.resource.IResourceVisitor;

public class ImageResource extends Resource {
    private final byte[] imageData;

    public ImageResource(String url, byte[] imageData) {
        super(url);
        this.imageData = imageData;
    }

    public byte[] getImageData() {
        return imageData;
```

```
    }

    @Override
    public void visit(IResourceVisitor visitor) throws Exception {
        visitor.visit(this);
    }
}
```

CssResourceFactory.java

```
package com.edu.web.restservicewebbrowser.domain.resource;

import com.edu.web.restservicewebbrowser.visitor.resource.IResourceVisitor;

public class CssResource extends Resource {

    private final String cssContent;

    public CssResource(String url, String cssContent) {
        super(url);
        this.cssContent = cssContent;
    }

    public String getCssContent() {
        return cssContent;
    }

    @Override
    public void visit(IResourceVisitor visitor) throws Exception {
        visitor.visit(this);
    }
}
```

Опис коду з використання патерну Chain of responsibility

Цей код демонструє архітектурне рішення, в якому асинхронний сервіс парсингу (ParsingServiceImpl) використовує комбінацію патернів проєктування для ефективної обробки веб-ресурсів.

Замість створення монолітного класу, який знає всі деталі створення та збереження кожного типу ресурсу, логіка була розділена: патерн Factory Method відповідає за поліморфне створення об'єктів, а патерн Visitor (підготовлений у

коді) — за їх подальшу обробку (наприклад, збереження в базу даних).

Учасники патерну "Factory Method"

Ця частина відповідає за генерацію правильних об'єктів ресурсів на основі вхідних даних (URL).

1. Клієнт (Client)
   - Клас: ParsingServiceImpl
   - Роль: Виступає ініціатором процесу. Його відповідальність — завантажити та проаналізувати HTML-сторінок (за допомогою бібліотеки Jsoup), визначити тип необхідного ресурсу (на основі розширення файлу) та інстанціювати відповідну фабрику.
   - Взаємодія: Клієнт не створює ресурси напряму через new. Він викликає загальний метод factory.fetchAndPackageResource(url), отримуючи назад абстрактний об'єкт Resource.
2. Абстрактний Творець (Creator)
   - Клас: ResourceFactory
   - Роль: Це абстрактний базовий клас, який визначає "контракт" для всіх фабрик.
   - Ключова деталь: Він містить абстрактний фабричний метод createResource(), реалізацію якого делегує підкласам. Також він містить спільну бізнес-логіку (fetchAndPackageResource), яка використовує цей фабричний метод для отримання готового продукту.
3. Конкретні Творці (Concrete Creators)
   - Класи: CssResourceFactory, JsResourceFactory, ImageResourceFactory
   - Роль: Це реальні фабрики, що містять специфічну логіку створення.
   - Взаємодія: Кожна фабрика реалізує метод createResource() і повертає екземпляр відповідного конкретного продукту: CssResource, JsResource або ImageResource.
4. Абстрактний Продукт (Product)
   - Клас: Resource
   - Роль: Це базовий клас для всіх типів ресурсів, що визначає спільний стан (наприклад, url) та інтерфейс для взаємодії (включаючи метод visit для Visitor).
5. Конкретні Продукти (Concrete Products)
   - Класи: CssResource, JsResource, ImageResource
   - Роль: Це кінцеві об'єкти, які створюються фабриками. Вони розширюють базовий клас Resource, додаючи специфічні дані (наприклад, cssContent для стилів або imageData для зображень).

Учасники патерну "Visitor" (Підготовка)

Код реалізує механізм "подвійної диспетчеризації" для майбутньої обробки створених ресурсів без модифікації їхніх класів.

1. Елемент (Element)
   - Клас: Resource (та його нащадки JsResource, ImageResource тощо).
   - Роль: Об'єкти, які можуть бути "відвідані". Абстрактний клас оголошує метод visit(IResourceVisitor).
   - Взаємодія: Кожен конкретний клас реалізує цей метод, викликаючи visitor.visit(this). Це дозволяє передати посилання на конкретний тип об'єкта (this у класі JsResource є типом JsResource), що дозволяє Візитору вибрати правильний метод перевантаження без використання instanceof.
2. Відвідувач (Visitor)
   - Інтерфейс: IResourceVisitor (використовується у сигнатурах методів).
   - Роль: Визначає інтерфейс для операцій, що виконуються над елементами.
   - Застосування: Хоча конкретна реалізація не наведена, архітектура передбачає створення класів на кшталт ResourceSaveVisitor. Такий візитор буде використовуватись у ResourceServiceImpl для розподілу логіки збереження кожного типу ресурсу у відповідні таблиці бази даних.

Переваги архітектурного підходу

Дана реалізація забезпечує високий рівень відокремлення (decoupling) компонентів:

1. ParsingServiceImpl (Клієнт) ізольований від логіки створення об'єктів завдяки Factory Method.
2. ResourceServiceImpl ізольований від специфічної логіки обробки/збереження кожного типу ресурсу завдяки Visitor.
3. Розширюваність: Додавання нового типу ресурсу (наприклад, FontResource) вимагатиме лише створення нового класу ресурсу, нової фабрики та оновлення інтерфейсу візитора, не порушуючи існуючий код парсингу.

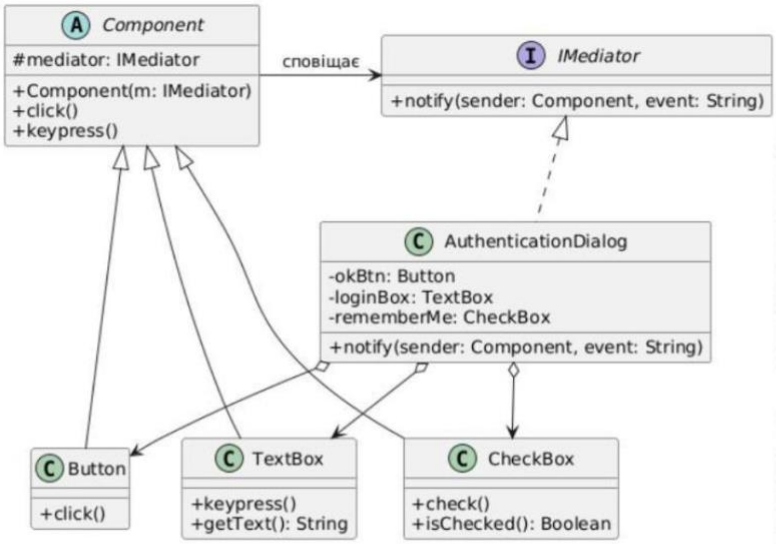
Контрольні запитання

1. Яке призначення шаблону «Посередник»?

Призначення шаблону «Посередник» полягає в тому, щоб зменшити хаотичні та прямі зв'язки між багатьма об'єктами в системі. Замість того, щоб кожен об'єкт

знав про всі інші та спілкувався з ними напряму (що створює заплутану павутину залежностей), вони спілкуються лише з одним центральним об'єктом — Посередником. Цей патерн інкапсулює логіку взаємодії, дозволяючи змінювати схему спілкування компонентів, не змінюючи самі компоненти. Це робить систему більш гнучкою та легкою для підтримки.

2. Нарисуйте структуру шаблону «Посередник».



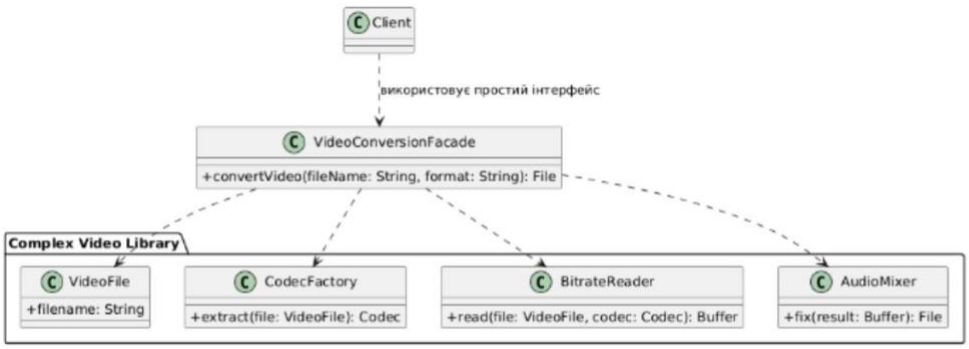
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

У цей шаблон входять чотири основні учасники. Перший — це інтерфейс **Посередника** (Mediator), який оголошує метод для отримання сповіщень від компонентів. Другий — це **Конкретний Посередник** (Concrete Mediator), який реалізує цей інтерфейс, зберігає посилання на всі підпорядковані йому компоненти та координує їхню роботу. Третій — це базовий клас **Компонента** (або Колеги), який зберігає посилання на об'єкт Посередника. Четвертий — це **Конкретні Компоненти** (Concrete Components), які виконують свою бізнес-логіку. Взаємодія відбувається так: коли в Компоненті стається подія, він не викликає інші компоненти напряму, а повідомляє про це Посередника. Посередник, отримавши сповіщення, вирішує, які саме інші компоненти потрібно задіяти, і викликає відповідні методи в них. Таким чином, компоненти нічого не знають один про одного.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» призначений для надання простого та уніфікованого інтерфейсу до складної системи класів, бібліотеки або фреймворку. Він діє як "парадний вхід", приховуючи від клієнта складність ініціалізації, залежностей та правильного порядку викликів численних об'єктів підсистеми. Головна мета — спростити використання системи для клієнта, якому не потрібна вся потужність та гнучкість низькорівневих налаштувань, а достатньо виконати типові сценарії.

5. Нарисуйте структуру шаблону «Фасад».



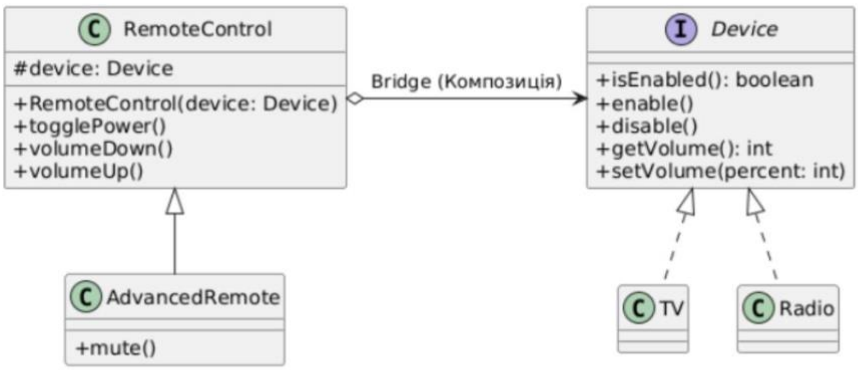
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

До шаблону входять два основні типи учасників. Перший — це власне клас **Фасаду** (Facade), який знає, яким класам підсистеми потрібно переадресувати запит клієнта і в якому порядку. Другий — це класи **Складної Підсистеми** (Subsystem classes), які виконують реальну корисну роботу. Взаємодія виглядає наступним чином: Клієнт викликає простий метод у Фасаду (наприклад, videoConverter.convert()). Фасад всередині цього методу виконує всю брудну роботу: створює об'єкти аудіо-кодеків, відео-кодеків, бітрейт-рідерів, налаштовує їх, зв'язує між собою і запускає процес. Класи підсистеми при цьому не знають про існування Фасаду і працюють так, ніби їх викликали напряму.

7. Яке призначення шаблону «Міст»?

Призначення шаблону «Міст» полягає у розділенні абстракції та її реалізації так, щоб вони могли змінюватися незалежно одна від одної. Цей патерн використовується для вирішення проблеми експоненційного зростання кількості класів при використанні успадкування, коли клас має кілька незалежних вимірів змін (наприклад, Форма: Коло/Квадрат і Колір: Червоний/Синій). «Міст» пропонує замінити жорстке успадкування на композицію, виділивши один із вимірів в окрему ієрархію класів.

8. Нарисуйте структуру шаблону «Міст».



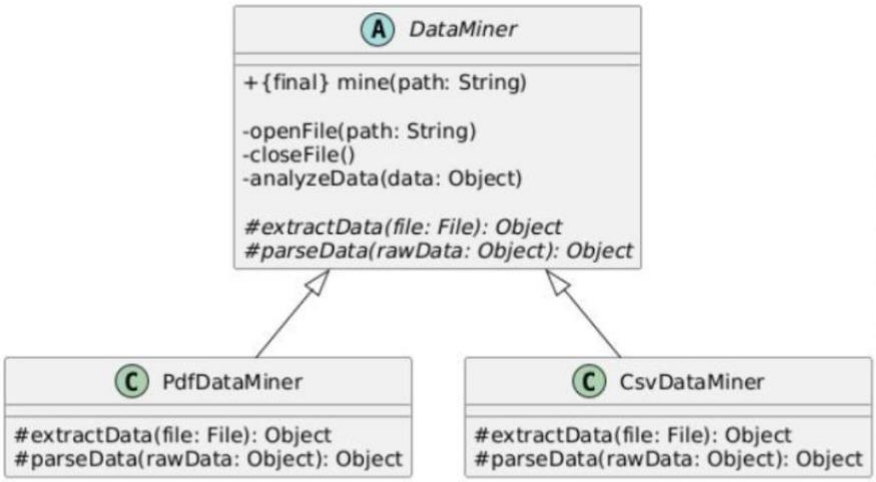
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

У шаблон входять чотири елементи. **Абстракція** (Abstraction) — це базовий клас високого рівня, який містить посилання на об'єкт реалізації і делегує йому роботу. **Уточнена Абстракція** (Refined Abstraction) — це підкласи абстракції, що розширюють її логіку (наприклад, клас Circle або Square). **Реалізація** (Implementation) — це інтерфейс для другої ієрархії (наприклад, інтерфейс Color). **Конкретна Реалізація** (Concrete Implementation) — це класи, що реалізують цей інтерфейс (наприклад, RedColor, BlueColor). Взаємодія полягає в тому, що Абстракція не виконує роботу сама, а перенаправляє виклики об'єкту Реалізації, який вона зберігає. Клієнт працює з Абстракцією, але може налаштувати її будь-якою Конкретною Реалізацією.

10. Яке призначення шаблону «Шаблонний метод»?

Призначення шаблону «Шаблонний метод» — визначити скелет алгоритму в базовому класі, делегуючи реалізацію деяких його кроків підкласам. Це дозволяє підкласам переопределяти певні етапи алгоритму без зміни його загальної структури та послідовності виконання. Це фундаментальний патерн для повторного використання коду, який дозволяє винести спільну частину алгоритму в батьківський клас, а варіативну залишити нащадкам.

11. Нарисуйте структуру шаблону «Шаблонний метод».



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

До шаблону входять **Абстрактний клас** та **Конкретні класи**. Абстрактний клас містить сам «шаблонний метод», який викликає набір кроків у певному порядку. Деякі кроки можуть бути вже реалізовані (базові операції), а деякі оголошені як абстрактні. Конкретні класи успадковуються від абстрактного і реалізують ці відсутні абстрактні кроки. Взаємодія проста: клієнт викликає шаблонний метод у об'єкта конкретного класу. Цей метод починає виконувати кроки алгоритму. Коли доходить черга до специфічного кроку, викликається реалізація, написана в конкретному класі-нащадку.

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

Головна відмінність полягає в меті та масштабі. «Фабричний метод» — це породжувальний патерн, який спеціалізується виключно на створенні об'єктів; він делегує підкласам рішення про те, який саме клас інстанціювати. «Шаблонний метод» — це поведінковий патерн, який керує алгоритмом дій; він делегує підкласам реалізацію частин цього алгоритму. Можна сказати, що Фабричний метод часто є окремим випадком Шаблонного методу, де "кроком", який треба реалізувати, є створення об'єкта.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає функціональність динамічного перемикання реалізації під час виконання програми та запобігає комбінаторному вибуху кількості класів. Він дозволяє розробляти ієрархію абстракцій (наприклад, види пультів керування) та ієрархію реалізацій (наприклад, драйвери різних телевізорів) абсолютно незалежно одна від одної. Якщо у вас є M видів пультів і N видів телевізорів, без Моста вам довелося б створити M*N класів. З Мостом ви створюєте лише M+N класів, і будь-який пульт може працювати з будь-яким телевізором, просто підставивши потрібний об'єкт реалізації.

Висновки

У ході виконання лабораторної роботи було закріплено навички проєктування архітектури програмних систем на прикладі веббраузера з використанням технологій JavaFX та Jakarta EE. Практична реалізація продемонструвала ефективність інтеграції різних груп патернів для вирішення архітектурних задач. Зокрема, застосування патерну Proxy дозволило повністю ізолювати клієнтський інтерфейс від складної логіки мережевої взаємодії, а Chain of Responsibility забезпечив побудову гнучкого конвеєра для послідовної обробки запитів на завантаження сторінок. На серверній стороні використання Factory Method та Template Method дозволило уніфікувати процеси створення та обробки різнотипних веб-ресурсів, винісши спільні алгоритми в абстракції та делегувавши специфічну логіку підкласам. У результаті було створено модульну та слабкозв’язану систему, яка відповідає принципам SOLID, є легкою у підтримці та відкритою до розширення функціоналу.