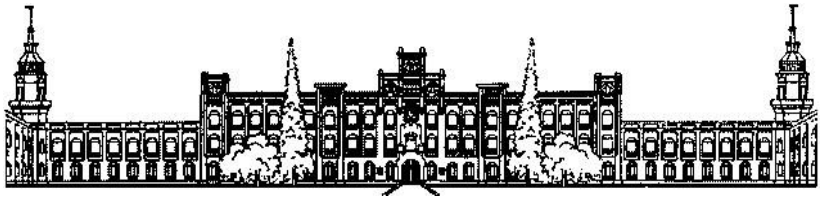


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

# Лабораторна робота №7

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав:

студент групи ІА-32

Нагорний Максим

Перевірив:

Мягкий М. Ю.

**Київ 2025**

# Зміст

# Вступ

**Тема:** Вступ до паттернів проектування.

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

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.

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

# Теоретичні відомості

## Шаблон «Mediator»

Шаблон проєктування Mediator належить до групи поведінкових патернів і використовується для організації взаємодії між об’єктами таким чином, щоб зменшити рівень їхньої залежності один від одного. У системах із великою кількістю об’єктів часто виникає проблема надмірної кількості прямих зв’язків, що ускладнює підтримку та модифікацію програми. Використання Mediator дозволяє сконцентрувати логіку взаємодії у спеціальному об’єкті-посереднику, який визначає, які компоненти повинні отримати повідомлення і як вони мають реагувати. Учасники системи не контактують один з одним безпосередньо, а взаємодіють лише через посередника, надсилаючи запити або повідомлення. Це дає змогу значно знизити зв’язність компонентів і зробити архітектуру більш гнучкою. Mediator спрощує модифікацію поведінки об’єктів, дозволяє легше змінювати схему взаємодії між ними та сприяє кращій підтримуваності коду, оскільки при додаванні нових об’єктів або зміні поведінки не треба змінювати інші елементи системи.

## Шаблон «Facade»

Шаблон Facade належить до структурних патернів і використовується для спрощення роботи зі складними підсистемами. У програмних системах часто є модулі або бібліотеки з великою кількістю функцій, які важко зрозуміти або які потребують виконання довгих послідовностей кроків. Facade створює простий, єдиний інтерфейс, що інкапсулює складну внутрішню логіку і приховує від користувача деталі реалізації. Завдяки цьому зовнішній код взаємодіє лише з фасадом, не маючи потреби розуміти або враховувати численні класи, залежності та їхню взаємодію. Це також знижує зв’язність між компонентами системи, оскільки зовнішні модулі не залежать від внутрішньої структури підсистеми. Використання фасаду підвищує зрозумілість коду, полегшує його супровід та дозволяє розробникам змінювати внутрішнє наповнення підсистеми без впливу на її користувачів. Крім того, такий патерн дає можливість додатково контролювати доступ до складних компонентів, обмежуючи можливості неправильного використання.

## Шаблон «Bridge»

Шаблон Bridge також належить до структурних патернів і призначений для відокремлення абстракції від її реалізації таким чином, щоб вони могли розвиватися незалежно одна від одної. У

традиційній моделі одна абстракція жорстко пов’язана з конкретною реалізацією, що ускладнює розширення системи або заміну механізмів роботи. Bridge пропонує поділити клас на дві ієрархії: абстракцію, яка містить високорівневу логіку, і реалізацію, яка містить низькорівневі операції. Абстракція містить посилання на об’єкт реалізації, і всі виклики делегуються йому. Така гнучка структура дозволяє створювати різні варіанти абстракцій і реалізацій, поєднуючи їх у будь-якій комбінації без потреби модифікувати старий код. Bridge забезпечує легше масштабування системи, підтримку платформи або зміну технологій, адже розробник може замінити або доповнити реалізацію без зміни абстракції. Це особливо корисно у великих системах, де складно передбачити всі варіанти поведінки або коли необхідно підтримувати різні конфігурації роботи.

**Шаблон «Template Method»**

Шаблон Template Method належить до поведінкових патернів і використовується для визначення загальної структури алгоритму в базовому класі, залишаючи його окремі кроки на реалізацію підкласами. У цьому патерні базовий клас містить так званий «шаблонний метод», який описує послідовність операцій, що формують алгоритм. Окремі кроки цього алгоритму можуть бути реалізовані у вигляді абстрактних або перевизначуваних методів, і підкласи надають їм konkretну поведінку. Завдяки цьому розробник може змінювати окремі частини алгоритму, не змінюючи його структуру в цілому. Це сприяє повторному використанню коду, адже загальна логіка зосереджена у базовому класі, а варіативність — у дочірніх. Template Method забезпечує контроль над порядком виконання операцій і дозволяє уникнути дублювання коду при реалізації схожих алгоритмів. Такий підхід особливо ефективний у ситуаціях, коли декілька класів мають виконувати однакові за структурою дії, але деякі кроки повинні виконуватися по-різному.

## Хід виконання

Для розробки системи архіватора було обрано шаблон проєктування «Facade». Вибір цього шаблону обумовлений необхідністю створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми

Суть шаблону «Фасад» полягає у тому, що створюється єдиний спрощений інтерфейс для виконання складної логіки, приховуючи деталі реалізації від клієнта. Таким чином, шаблон було застосовано для зняття навантаження та спрощення контролера

Для реалізації цього шаблону було створено загальний клас ArchiveFactory, цей клас відповідає за обробку завантажених файлів, генерації чексаму, формування шляху до архіву, виклик сервісу, для вибору стратегії архівації, розбиття на частини, збереження архіву в БД.

Переваги обраного шаблону:

- Спрощена робота контролера;
- Легкість підтримання та доповнення коду;
- підтримка SOLID;
- інкапсуляція внутрішньої структури від клієнтського коду;

На рис. 1 наведена модифікована діаграма класів із представленням використання шаблону в реалізації системи

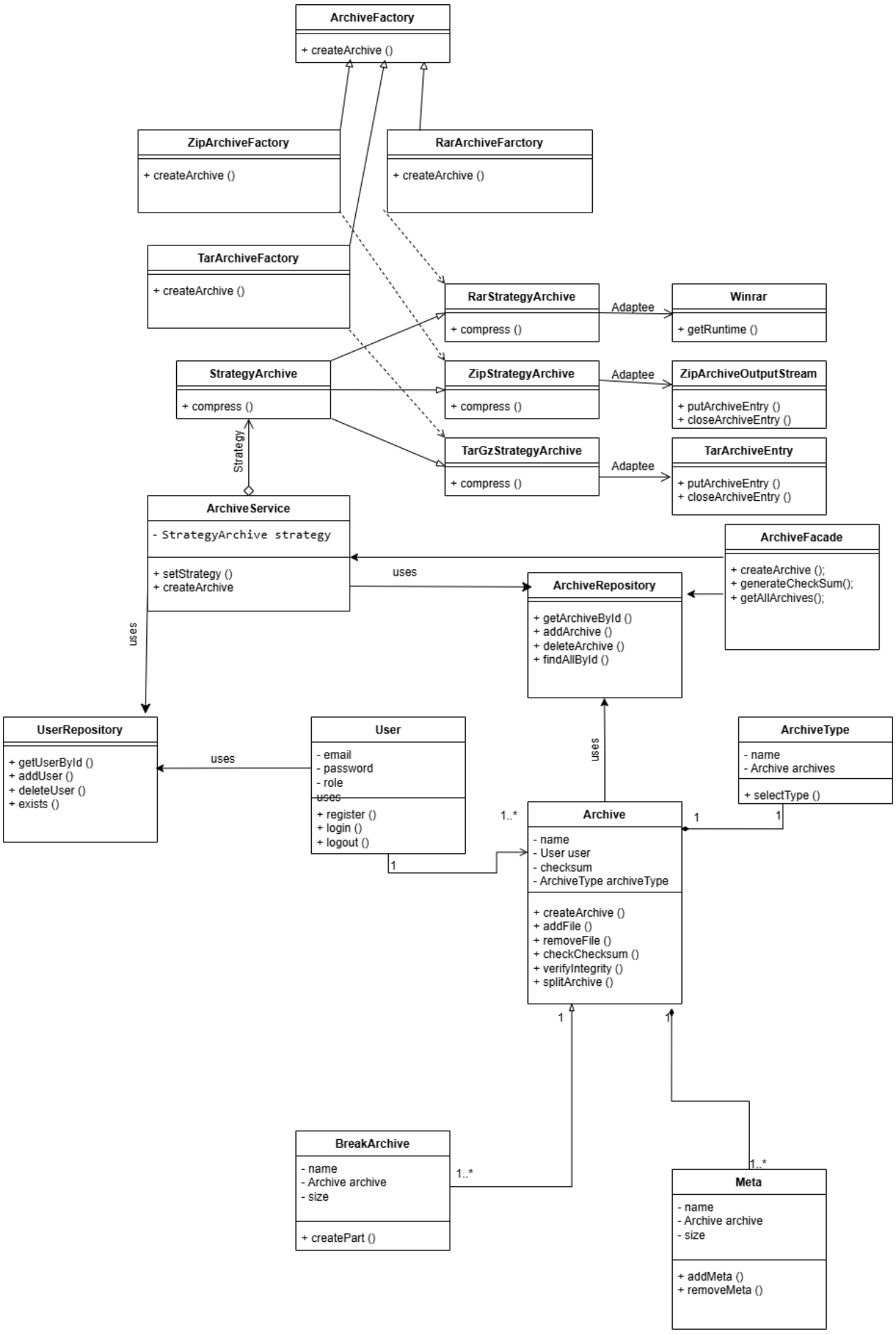



*Рис. 1. Діаграма класів*

## Код програми

```
package com.example.demo.facade;
import com.example.demo.model.Archive;
import com.example.demo.model.Meta;
import com.example.demo.repository.ArchiveRepository;
import com.example.demo.service.ArchiveService;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;
import java.io.File;
import java.io.IOException;
import java.security.MessageDigest;
import java.util.*;
@Component
public class ArchiveFacade {
    private final ArchiveService archiveService;
    private final ArchiveRepository archiveRepository;
    public ArchiveFacade(ArchiveService archiveService, ArchiveRepository archiveRepository)
{
        this.archiveService = archiveService;
        this.archiveRepository = archiveRepository;
    }
    public void createArchive(Archive archive, MultipartFile[] files) throws IOException {
        List<File> tempFiles = new ArrayList<>();
        Set<Meta> metaSet = new HashSet<>();
        for (MultipartFile multipart : files) {
            if (!multipart.isEmpty()) {
                File temp = File.createTempFile("upload_", "_" + multipart.getOriginalFilename());
                multipart.transferTo(temp);
                tempFiles.add(temp);
                Meta meta = new Meta();
                meta.setFileName(multipart.getOriginalFilename());
                meta.setSize(multipart.getSize());
                metaSet.add(meta);
            }
```

```
        }
        archive.setMeta(metaSet);
        archive.setChecksum(generateChecksum(archive));
        String outputDir = "C:/archives/";
        new File(outputDir).mkdirs();
        String outputPath = outputDir + archive.getName() + "." + archive.getArchiveTypeId();
        try {
            archiveService.createArchive(tempFiles, outputPath, archive.getArchiveTypeId());
        } finally {
            for (File f : tempFiles) f.delete();
        }
        archive.setBreakArchiveIds(Set.of("part1", "part2"));
        archiveRepository.save(archive);
    }
    private String generateChecksum(Archive arch) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            String data = arch.getName() + arch.getArchiveTypeId() + System.currentTimeMillis();
            byte[] hash = digest.digest(data.getBytes());
            StringBuilder hex = new StringBuilder();
            for (byte b : hash) hex.append(String.format("%02x", b));
            return hex.toString();
        } catch (Exception e) {
            return "error";
        }
    }
    public List<Archive> getAllArchives() {
        return archiveRepository.findAll();
    }
}
```

## Висновок

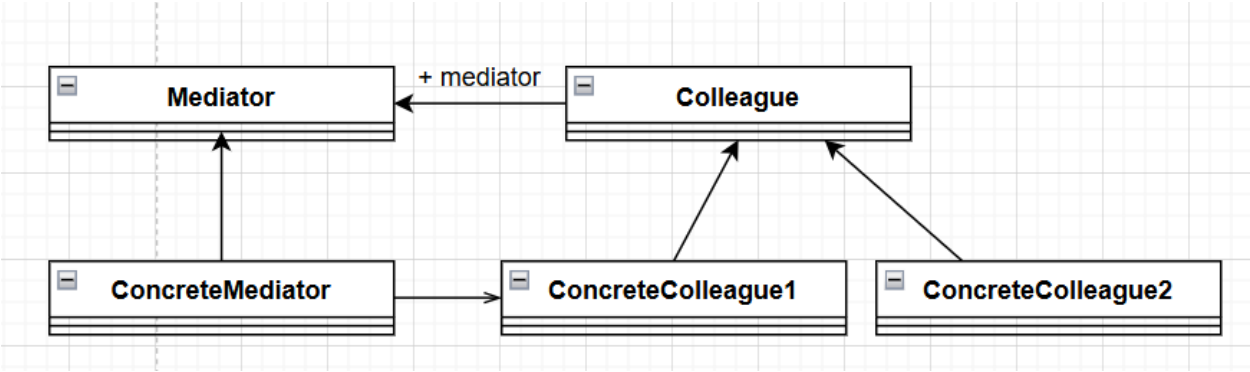
Під час виконання лабораторної роботи було реалізовано шаблон «Facade» згідно теми «архіватор», під час реалізації було розроблено загальний єдиний спрощений клас для виконання складної логіки, приховуючи деталі реалізації від клієнта ArchiveFacade, що відповідає за обробку завантажених файлів, генерації чексаму, формування шляху до архіву, виклик сервісу, для вибору стратегії архівації, розбиття на частини, збереження архіву в БД.

## Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Шаблон «Посередник» призначений для зменшення кількості прямих зв'язків між об'єктами, які повинні взаємодіяти між собою. Замість того щоб кожен об'єкт контактував безпосередньо з іншими, комунікація концентрується у спеціальному об'єкті-посереднику, який контролює передачу повідомлень, координує поведінку і забезпечує слабку зв'язаність компонентів. Це полегшує підтримку й модифікацію системи, оскільки зміни у взаємодії реалізуються лише в посереднику, а не у всіх класах.

2. Нарисуйте структуру шаблону «Посередник».




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

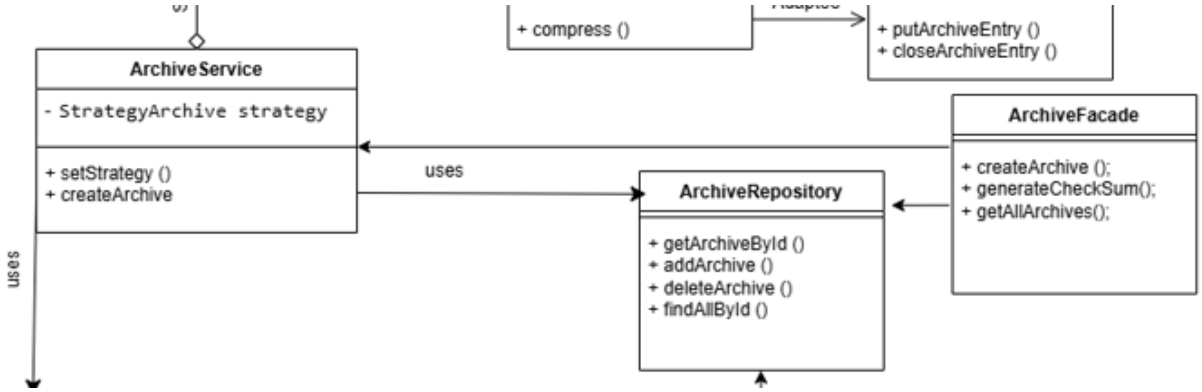
До шаблону входять абстрактний посередник, конкретний посередник, абстрактні колеги та конкретні колеги. Колеги не взаємодіють один з одним напряму, вони передають інформацію посереднику, який, у свою чергу, визначає, кому саме слід відправити повідомлення. Взаємодія повністю контролюється конкретним посередником, який управляє обміном даними між об'єктами.

4. Яке призначення шаблону «Фасад»?

Шаблон «Фасад» призначений для спрощення доступу до складної підсистеми шляхом надання єдиного спрощеного інтерфейсу. Він приховує складну структуру внутрішніх класів, скорочує кількість необхідних кроків

для роботи з підсистемою і забезпечує зручний спосіб взаємодії. Фасад зменшує залежність клієнтського коду від деталей реалізації й робить систему легше керованою та зрозумілою.

5. Нарисуйте структуру шаблону «Фасад».



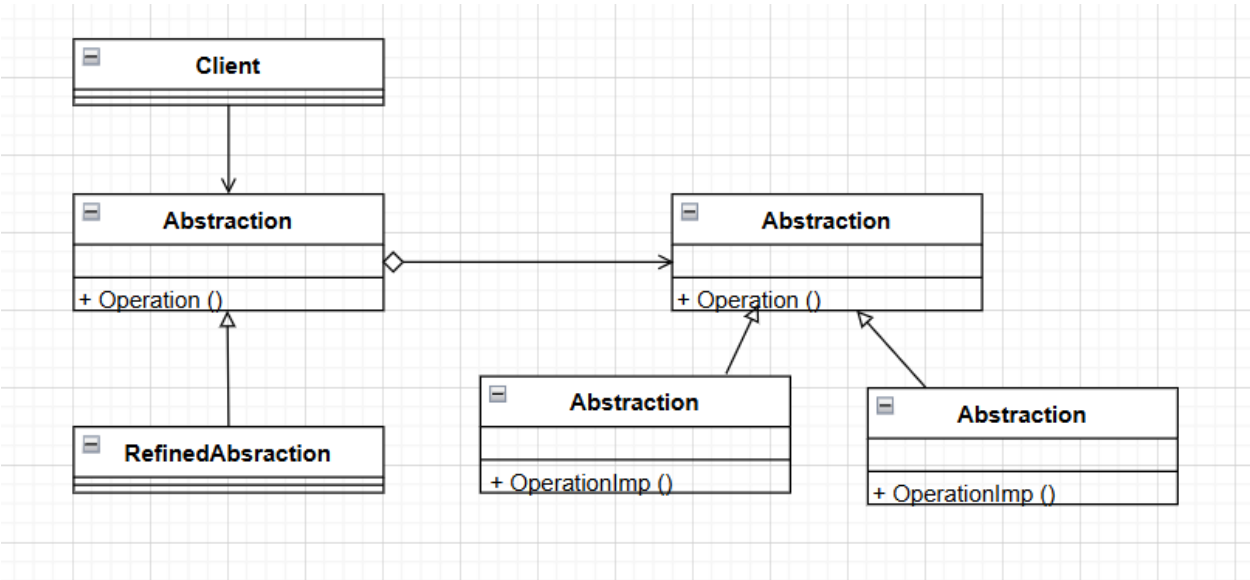

6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

У шаблоні є клас-фасад і класи підсистеми, які виконують фактичну роботу. Клієнт взаємодіє лише з фасадом, який делегує виклики внутрішнім компонентам. Класи підсистеми можуть взаємодіяти між собою незалежно від фасаду, але клієнт не має до них прямого доступу. Фасад координує роботу підсистеми і визначає порядок виконання операцій.

7. Яке призначення шаблону «Міст»?

Шаблон «Міст» призначений для відокремлення абстракції від її реалізації, щоб вони могли змінюватися незалежно одна від одної. Він дозволяє поєднувати різні варіанти абстракцій з різними варіантами реалізацій без створення великої кількості підкласів. Це забезпечує гнучкість, масштабованість та спрощує розширення системи.

8. Нарисуйте структуру шаблону «Міст».



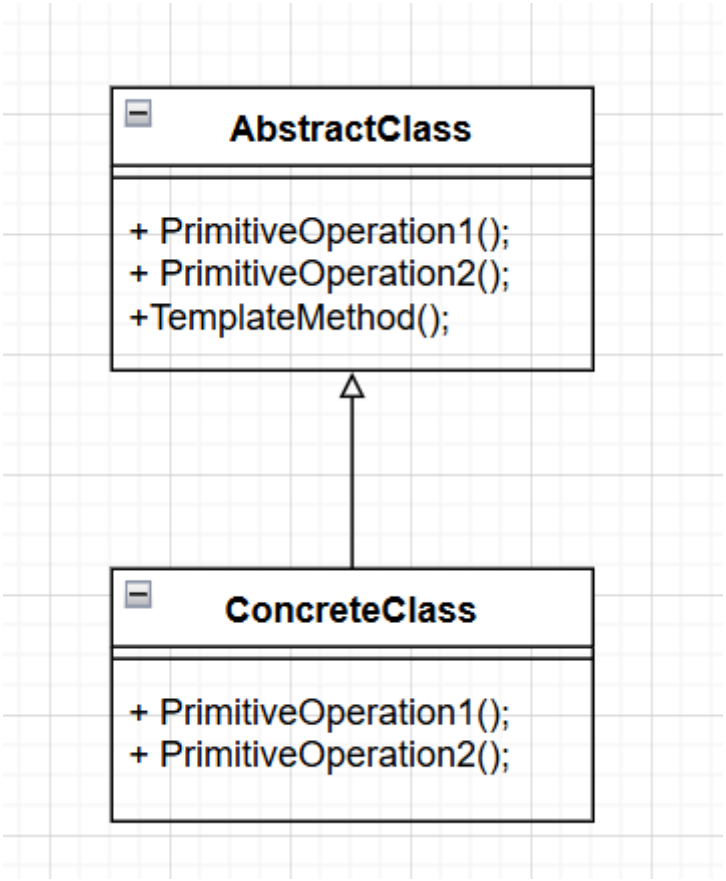
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

До шаблону входять абстракція, розширена абстракція, інтерфейс реалізації та конкретні реалізатори. Абстракція містить посилання на реалізацію і делегує їй низькорівневі операції. Конкретні реалізатори визначають специфічну поведінку, а розширені абстракції можуть додавати додаткову логіку. Взаємодія побудована так, щоб зміна абстракції не вимагала зміни реалізації, і навпаки.

10. Яке призначення шаблону «Шаблоний метод»?

Шаблонний метод призначений для визначення загальної структури алгоритму у базовому класі, дозволяючи підкласам змінювати окремі його кроки, не порушуючи загальної послідовності. Він забезпечує перевикористання спільної логіки, контроль порядку виконання і можливість варіації в деталях алгоритму. Це дозволяє легко створювати різні модифікації поведінки з єдиною структурою.

11. Нарисуйте структуру шаблону «Шаблоний метод».




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

До шаблону входять базовий клас із шаблонним методом та один або кілька підкласів, які реалізують окремі етапи алгоритму. Базовий клас викликає методи, які можуть бути реалізовані або перевизначені в підкласах.

Підкласи забезпечують конкретну поведінку, а базовий клас контролює структуру послідовності виконання.

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

Шаблонний метод визначає структуру цілого алгоритму, залишаючи частину кроків на реалізацію підкласів, тобто управляє алгоритмічним процесом. Фабричний метод, навпаки, фокусується на створенні об’єктів і дозволяє підкласам визначати, який саме об’єкт створювати. Головна різниця полягає в тому, що шаблонний метод регулює хід виконання алгоритму, а фабричний метод вирішує питання інстанціювання об’єктів.

14. Яку функціональність додає шаблон «Міст»?

Шаблон «Міст» додає можливість незалежно розвивати абстракції та їхні реалізації, зменшує кількість підкласів, забезпечує гнучке поєднання різних реалізацій з різними абстракціями та спрощує підтримку системи. Він усуває жорстке зв’язування між високорівневими та низькорівневими компонентами, що значно підвищує розширюваність та адаптивність програмної архітектури.