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 розроблення програмного забезпечення

Тема проєкту ‘Shell (total commander)’

Виконала: студентка групи ІА-33

Маляревич Анжела Валентинівна

Перевірив:

Мягкий Михайло Юрійович

Київ - 2025

Зміст

Тема: Патерни проектування.

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

Тема роботи:

18. Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server) Оболонка повинна вміти виконувати основні дії в системі – перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

Теоретичні відомості

Mediator (Посередник)

Патерн вводить окремий об’єкт-координатор, який керує усіма взаємодіями між компонентами. Замість того, щоб класи напряму звертались один до одного, вони працюють через медіатора. Це зменшує кількість зв’язків, спрощує підтримку та дає можливість легко змінювати логіку взаємодій. Використовується там, де багато елементів впливають один на одного.

Facade (Фасад)

Фасад надає один спрощений інтерфейс до великої і складної підсистеми. Зовнішній код працює тільки з фасадом і не бачить внутрішньої складності. Завдяки цьому система легше у використанні, а внутрішні зміни не зачіпають клієнтів. Підходить, коли потрібно «сховати» багато класів за одним доступним API.

Bridge (Міст)

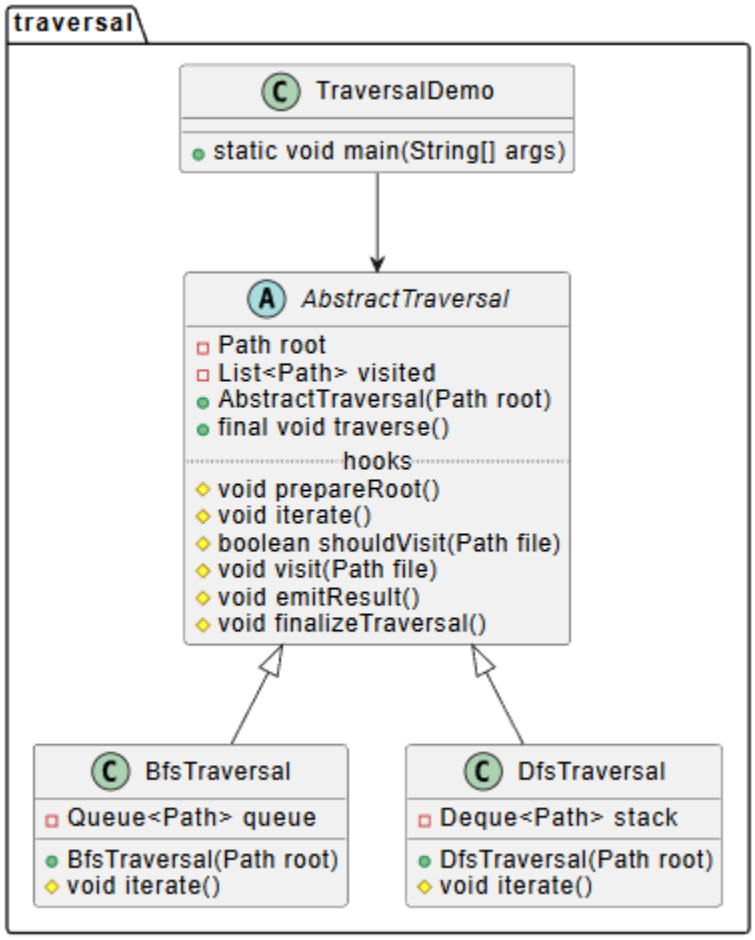
Bridge розділяє дві незалежні частини системи: абстракцію (наприклад, фігури) і реалізацію (способи відображення). Абстракція містить посилання на об’єкт реалізації й делегує йому роботу. Це дозволяє окремо додавати нові фігури та нові способи відображення без перехресного множення класів. Дає гнучкість і чистішу архітектуру.

Template Method (Шаблонний метод)

У базовому класі описується загальний алгоритм, а окремі кроки робляться методами, які підкласи перевизначають. Це корисно, коли алгоритми схожі, але мають відмінності в деталях. Більша частина логіки лишається спільною, а специфічні дії реалізуються в дочірніх класах, що зменшує дублювання коду.

Хід роботи

Діаграма класів



Абстрактний шаблон обходу

Template Method — фіксує скелет алгоритму обходу.

```
public final void traverse() throws IOException {  2 usages
    prepareRoot();
    iterate();
    emitResult();
    finalizeTraversal();
}
```

```
    protected void prepareRoot() throws IOException {  3 usages  2 overrides
        if (!Files.exists(root)) {
            throw new NoSuchFileException(root.toString());
        }
    }

    protected abstract void iterate() throws IOException;  1 usage  2 implementations


    protected boolean shouldVisit(Path path) {                                  2 usag
        return true;
    }


    protected void visit(Path path) {  2 usages
        visited.add(path);
    }


    protected void emitResult() {  3 usages  2 overrides
        System.out.println("Visited " + visited.size() + " items:");
        visited.forEach(System.out::println);
    }

    protected void finalizeTraversal() {  1 usage

    }

    protected List<Path> listChildren(Path dir) throws IOException {  2 usages
        if (!Files.isDirectory(dir)) return List.of();
        try (var stream = Files.list(dir)) {
            return stream.toList();
        }
    }
}
```

Абстрактний клас AbstractTraversal, який реалізує шаблонний метод traverse() та визначає кістяк алгоритму обходу файлової системи.

Реалізація обходу в ширину (BFS)

```java
public class BfsTraversal extends AbstractTraversal {  1 usage

    private final Queue<Path> queue = new ArrayDeque<>();  5 u

    public BfsTraversal(Path root) {  1 usage
        super(root);
    }

    @Override  no usages
    protected void prepareRoot() throws IOException {
        super.prepareRoot();
        queue.clear();
        queue.add(root);
    }

    @Override  no usages
    protected void iterate() throws IOException {
        while (!queue.isEmpty()) {
            Path current = queue.remove();

            if (!shouldVisit(current)) {
                continue;
            }

            visit(current);

            for (Path child : listChildren(current)) {
                queue.add(child);
            }
        }
    }

    @Override  no usages
    protected void emitResult() {
        System.out.println("BFS traversal from " + root);
        super.emitResult();
    }
}
```

Клас BfsTraversal, який наслідує AbstractTraversal і перевизначає крок iterate() для реалізації обходу файлової системи в ширину (BFS)

Реалізація обходу в глибину (DFS)

```java
public class DfsTraversal extends AbstractTraversal {  1 usage

    private final Deque<Path> stack = new ArrayDeque<>();  5 usages

    public DfsTraversal(Path root) {  1 usage
        super(root);
    }

    @Override  no usages
    protected void prepareRoot() throws IOException {
        super.prepareRoot();
        stack.clear();
        stack.push(root);
    }

    @Override  no usages
    protected void iterate() throws IOException {
        while (!stack.isEmpty()) {
            Path current = stack.pop();

            if (!shouldVisit(current)) {
                continue;
            }

            visit(current);

            // додаємо дітей у стек — вийде обхід у глибину
            var children = listChildren(current);
            for (Path child : children) {
                stack.push(child);
            }
        }
    }

    @Override  no usages
    protected void emitResult() {
        System.out.println("DFS traversal from " + root);
        super.emitResult();
    }
}
```

Клас DfsTraversal, який реалізує альтернативний варіант кроку iterate() для обходу файлової системи в глибину (DFS) на основі того ж шаблонного методу traverse()

Висновок

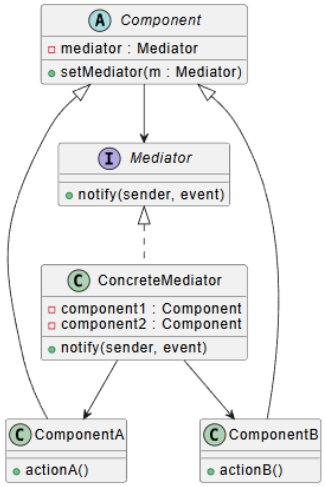
У цій лабораторній я застосувала патерн Template Method для організації спільного алгоритму обходу файлової системи. Я винесла основні кроки обходу в абстрактний метод traverse(), а різні варіанти обходу (BFS та DFS) реалізувала в окремих підкласах через перевизначення лише потрібних кроків. Це дозволило уникнути дублювання коду та зробило алгоритм гнучким і легко розширюваним. Мета роботи досягнута.

Питання до лабораторної роботи

1. Яке призначення шаблону «Посередник»?

Патерн «Посередник» призначений для керування взаємодією між багатьма об’єктами, щоб вони не спілкувалися напряму. Усі комунікації проходять через єдиний об’єкт-посередник, що зменшує зв’язність та спрощує модифікацію поведінки системи

2. Нарисуйте структуру шаблону «Посередник».



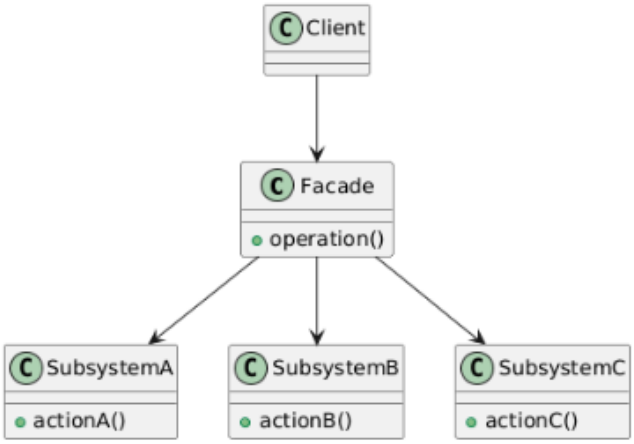
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

Mediator — інтерфейс для комунікації між компонентами.
ConcreteMediator — координує взаємодію між конкретними компонентами.
Component — базовий клас, що містить посилання на Mediator.
ConcreteComponentX — компоненти, які виконують дії, надсилаючи події посереднику.

4. Яке призначення шаблону «Фасад»?

Патерн «Фасад» спрощує роботу з складною підсистемою, надаючи єдиний, зручний, високорівневий інтерфейс. Клієнт працює з фасадом і не знає про складні внутрішні класи

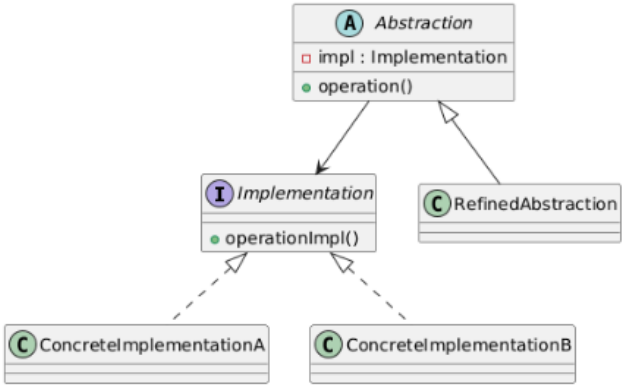
5. Нарисуйте структуру шаблону «Фасад».



6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?
Facade — єдина точка доступу до складної системи.
Subsystem classes (A, B, C) — внутрішні частини підсистеми.
Client — працює тільки через фасад.

7. Яке призначення шаблону «Міст»?
Патерн «Міст» відділяє абстракцію від реалізації, дозволяючи змінювати їх незалежно одна від одної. Ідеально підходить, коли є багато варіацій класів і потрібно уникнути вибуху ієрархій

8. Нарисуйте структуру шаблону «Міст».

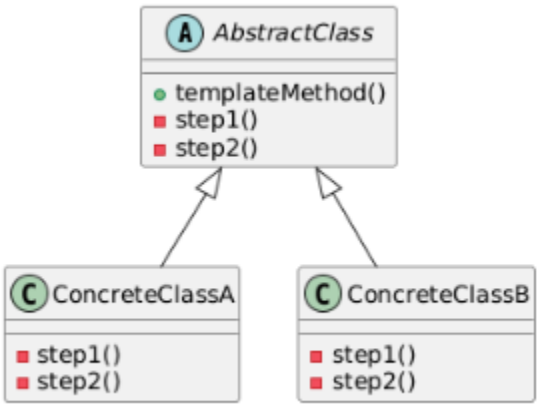


9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?
Abstraction — абстракція, яку використовує клієнт

RefinedAbstraction — уточнена абстракція
Implementation — інтерфейс реалізації
ConcreteImplementationX — конкретні реалізації

10.Яке призначення шаблону «Шаблоний метод»?
Визначити скелет алгоритму в базовому класі, дозволивши підкласам перевизначати окремі кроки алгоритму, не змінюючи його структури

11.Нарисуйте структуру шаблону «Шаблоний метод»



12.Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?
AbstractClass — визначає шаблонний метод (templateMethod()) та кроки алгоритму.
ConcreteClassX — реалізують конкретні кроки алгоритму.

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

Шаблонний метод задає алгоритм, а підкласи змінюють лише окремі його кроки. Фабричний метод визначає як створювати об’єкти, і підкласи вирішують, який саме об’єкт повернути.
Перший про поведінку, другий про створення об’єктів.

14.Яку функціональність додає шаблон «Міст»?

Патерн «Міст» дозволяє розділити абстракцію та реалізацію, щоб їх можна було змінювати незалежно. Додає гнучкість і дозволяє уникнути величезних ієрархій класів