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 №8

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 «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» 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

Composite (Компонувальник)

Дозволяє представляти об’єкти у вигляді дерева: прості елементи (Leaf) і складні, що містять інші елементи (Composite).
Клієнт працює з ними однаково. Зручно для ієрархій. Наприклад, проєкти → функції → user story → задачі.

Flyweight (Легковаговик)

Економить пам’ять, коли об’єктів дуже багато.
Розділяє стан на:

- внутрішній — спільний для всіх,
- зовнішній — задається під час використання.
  Наприклад: одна літера ‘а’, а позиції в тексті передаються окремо.

Interpreter (Інтерпретатор)

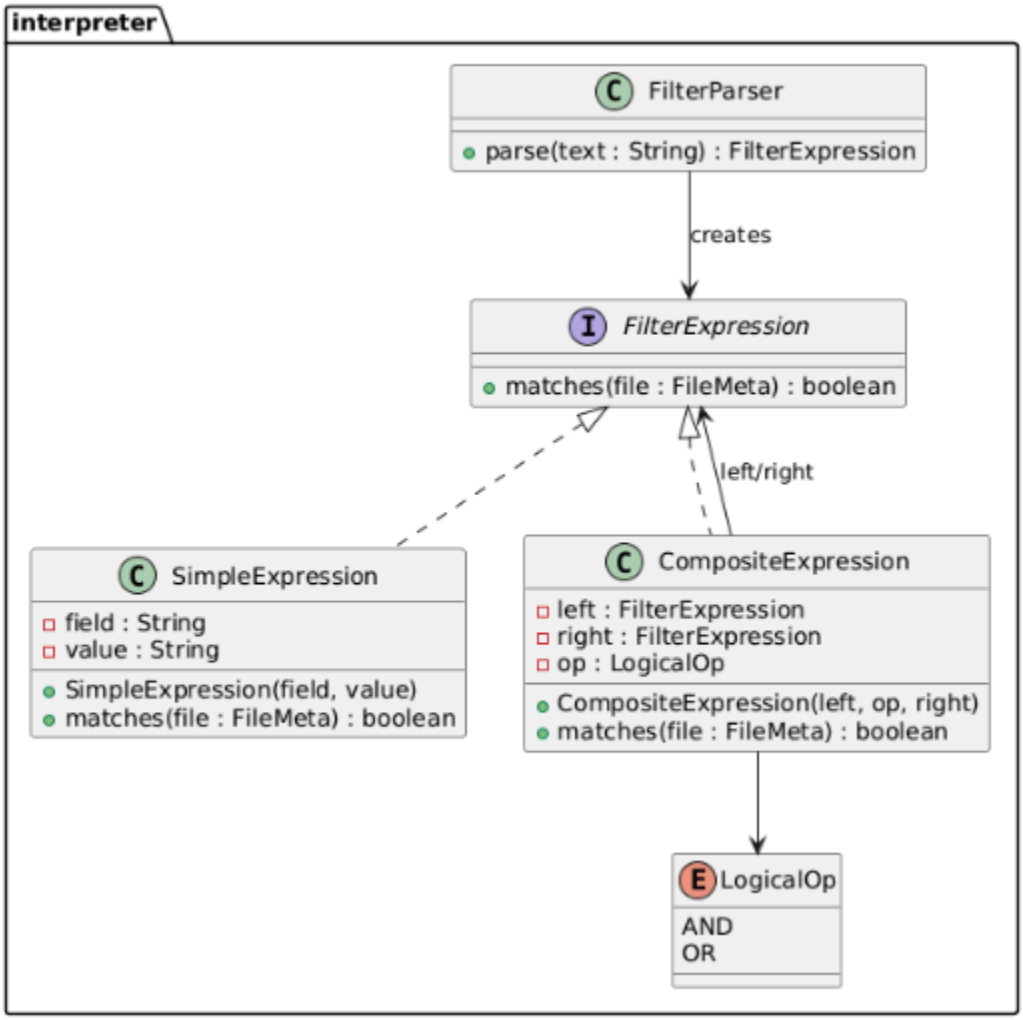
Використовується для створення простої мови та її обчислення.
Граматика подається як дерево виразів (термінальні та нетермінальні).
Кожен вузол вміє себе інтерпретувати. Застосовується, коли мова маленька.

Visitor (Відвідувач)

Дозволяє додавати нові операції до об’єктів без зміни їхніх класів.
Елемент приймає відвідувача, а той виконує потрібну дію.
Приклад: різні способи порахувати вартість товарів у кошику.

Хід роботи

Діаграма класів



На діаграмі показана структура реалізації патерну Interpreter у проєкті файлової оболонки. Інтерфейс FilterExpression є базовим елементом і визначає метод matches(), який виконує перевірку відповідності окремого результату пошуку умові фільтрації

Термінальний вираз SimpleExpression

```java
package ua.kpi.ia33.shellweb.patterns.search.interpreter;

import ua.kpi.ia33.shellweb.domain.SearchResult;


public class SimpleExpression implements FilterExpression {  1 usage

    private final String field;  2 usages
    private final String value;   4 usages

    public SimpleExpression(String field, String value) {  1 usage
        this.field = field.toLowerCase();
        this.value = value;
    }

    @Override  4 usages
    public boolean matches(SearchResult file) {
        return switch (field) {
            case "name" -> matchName(file);
            case "ext", "extension" -> matchExt(file);
            case "type" -> matchType(file);
            default -> false;
        };
    }

    private boolean matchName(SearchResult file) {  1 usage
        String name = file.getName() == null ? "" : file.getName().toLowerCase();
        String pattern = value.toLowerCase();

        if (pattern.startsWith("*") && pattern.endsWith("*") && pattern.length() > 2) {
            String inner = pattern.substring(1, pattern.length() - 1);
            return name.contains(inner);
        } else if (pattern.startsWith("*")) {
            String suffix = pattern.substring( beginIndex: 1);
            return name.endsWith(suffix);
        } else if (pattern.endsWith("*")) {
            String prefix = pattern.substring(0, pattern.length() - 1);
            return name.startsWith(prefix);
        } else {
            return name.equals(pattern);
        }
    }
```

Клас SimpleExpression представляє термінальні вирази – найпростіші предикати, такі як name:*pattern* або ext:pdf.
Клас CompositeExpression є нетермінальним виразом і поєднує декілька підвиразів за допомогою логічних операторів AND та OR, що описані в переліку LogicalOp

Парсер розширеного виразу фільтрації

```
private FilterExpression parseOr(ParserState s) {  2 usages
    FilterExpression expr = parseAnd(s);
    while (s.peekIs( expected: "OR")) {
        s.consume();
        FilterExpression right = parseAnd(s);
        expr = new CompositeExpression(expr, LogicalOp.OR, right);
    }
    return expr;
}

private FilterExpression parseAnd(ParserState s) {  2 usages
    FilterExpression expr = parsePrimary(s);
    while (s.peekIs( expected: "AND")) {
        s.consume();
        FilterExpression right = parsePrimary(s);
        expr = new CompositeExpression(expr, LogicalOp.AND, right);
    }
    return expr;
}

private FilterExpression parsePrimary(ParserState s) {  2 usages
    if (s.peekIs( expected: "(")) {
        s.consume();
        FilterExpression inner = parseOr(s);
        s.expect( expected: ")");
        return inner;
    }
    return parsePredicate(s);
}

private FilterExpression parsePredicate(ParserState s) {  1 usage
    String token = s.consume();
    String[] parts = token.split( regex: ":",  limit: 2);
    if (parts.length != 2) {
        throw new IllegalArgumentException("Невірний предикат: " + token);
    }
    String field = parts[0].trim();
    String value = parts[1].trim();
    return new SimpleExpression(field, value);
}
```

Клас FilterParser відповідає за розбір текстового запиту користувача, розділення його на токени та побудову відповідного дерева виразів (FilterExpression). Саме це дерево застосовується під час виконання пошуку файлів та фільтрації результатів.

```java
package ua.kpi.ia33.shellweb.patterns.search.interpreter;

import ua.kpi.ia33.shellweb.domain.SearchResult;

public class CompositeExpression implements FilterExpression {  2 usages

    private final FilterExpression left;  3 usages
    private final FilterExpression right;  3 usages
    private final LogicalOp op;  2 usages

    public CompositeExpression(FilterExpression left, LogicalOp op, FilterExpression right) {  2 usages
        this.left = left;
        this.op = op;
        this.right = right;
    }

    @Override  4 usages
    public boolean matches(SearchResult file) {
        return switch (op) {
            case AND -> left.matches(file) && right.matches(file);
            case OR  -> left.matches(file) || right.matches(file);
        };
    }
}
```

Клас FilterParser, що перетворює текстовий вираз у дерево об’єктів FilterExpression згідно з простою граматикою:
OrExpr := AndExpr ('OR' AndExpr)*
AndExpr := Primary ('AND' Primary)*
Primary := Predicate | '(' OrExpr ')'

Висновок

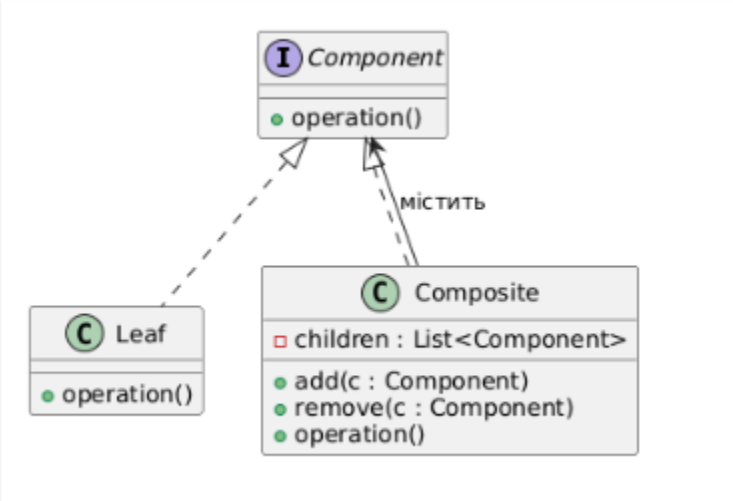
У ході виконання восьмої лабораторної роботи я реалізувала шаблон Interpreter у програмному комплексі файлової оболонки. Я створила власну міні-мову для фільтрації результатів пошуку, побудувала структуру термінальних і нетермінальних виразів та розробила парсер, який перетворює текстовий запит у дерево об’єктів. Реалізований інтерпретатор дозволяє комбінувати умови через AND, OR та дужки, що робить пошук значно гнучкішим. Виконана робота показала, як патерн Interpreter допомагає відокремити синтаксис мови від її виконання та спрощує розширення функціональності системи

Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

Дозволяє єдиним способом працювати з окремими об’єктами та їхніми деревоподібними групами. Тобто клієнт може однаково обробляти одиночний елемент і цілу ієрархію

2. Нарисуйте структуру шаблону «Композит».



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

Component — спільний інтерфейс для листків і вузлів (operation).

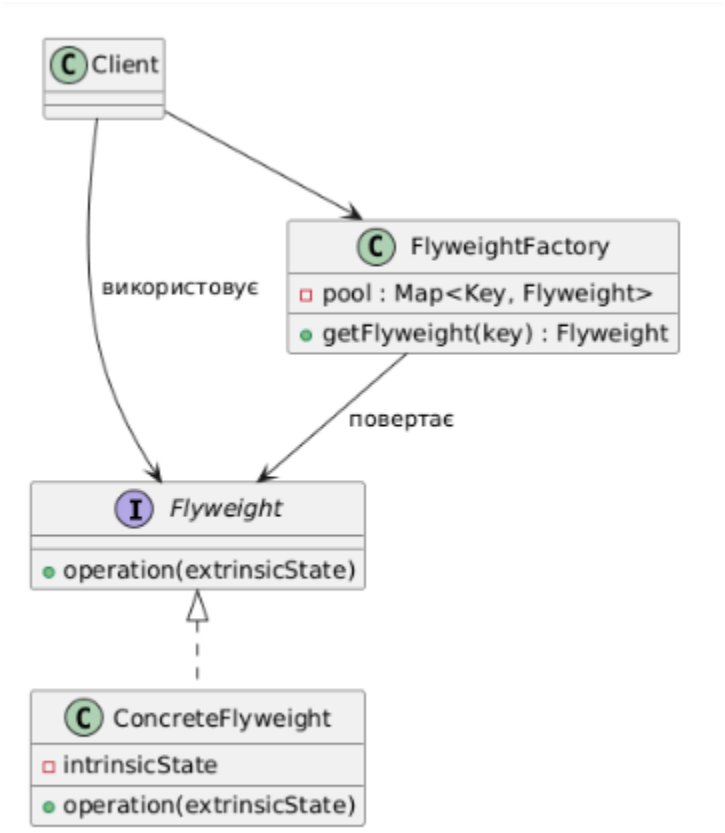
Leaf — простий елемент без нащадків.

Composite — вузол, який зберігає дочірні Component та делегує їм виклики

4. Яке призначення шаблону «Легковаговик»?

Зменшити витрати пам’яті, коли є дуже багато однотипних об’єктів, шляхом винесення спільного стану в один легковаговий об’єкт і розділення його між багатьма екземплярами

5. Нарисуйте структуру шаблону «Легковаговик».



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

Flyweight — інтерфейс, що приймає зовнішній стан (extrinsicState)

ConcreteFlyweight — зберігає внутрішній спільний стан (intrinsicState)

FlyweightFactory — повертає (або повторно використовує) екземпляри Flyweight

Client — зберігає зовнішній стан і звертається до FlyweightFactory

Фабрика слідкує, щоб однакові внутрішні стани не дублювалися

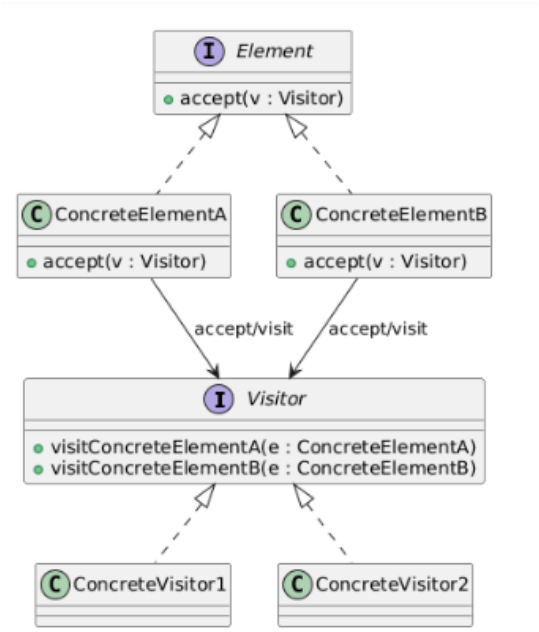
7. Яке призначення шаблону «Інтерпретатор»?

Описати просту мову або граматику у вигляді класів і забезпечити інтерпретацію виразів цієї мови, будуючи дерево об’єктів і обчислюючи результат через метод interpret()

8. Яке призначення шаблону «Відвідувач»?

Дозволяє додавати нові операції над об’єктами складної структури (дерево, композит), не змінюючи їх класи, за рахунок винесення операцій у окремі об’єкти-відвідувачі

9. Нарисуйте структуру шаблону «Відвідувач»



10.Які класи входять в шаблон «Відвідувач», та яка між ними взаємодія?

Visitor — інтерфейс з методами visitX() для кожного типу елементів.

ConcreteVisitorX — реалізують конкретні операції над елементами.

Element — інерфейс із методом accept(Visitor).

ConcreteElementX — приймають відвідувача й викликають для себе відповідний visitX(this).

Клієнт обходить структуру елементів, кожен елемент викликає accept(visitor), а Visitor виконує потрібну дію