



Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №8
Технології розроблення програмного забезпечення
«Патерни проектування. Паттерн «Visitor»»
«10. VCS all-in-one»

Виконав
студент групи ІА–32:
Варивода К. С.

Перевірив:
Мякий Михайло Юрійович

Зміст

Зміст.....	2
Варіант.....	2
Теоретичні відомості.....	3
Хід роботи.....	4
Висновок.....	6
Відповіді на питання	8

Варіант

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно.

Теоретичні відомості

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальноживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Призначення: Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів [6]. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача).

Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

Рішення: Основна ідея патерна «Відвідувач» це рознести логіку і дані в різні класи та ієрархії. Якщо так зробити для нашої онлайн-корзини, то в класі відвідувача ми маємо функції для обрахунку логіки для кожного типу, а об'єкти в корзині знають свій тип, отримують екземпляр відвідувача і в нього викликають метод відповідно до свого типу. В результаті ми робимо різні відвідувачі для розрахунку вартості: один для звичайних розрахунків, інший для розрахунку вартості зі знижкою, ще один для розрахунку сезонних знижок.

Релевантність до проекту:

Патерн Відвідувач (Visitor) є вкрай корисним для мого проєкту, оскільки дозволяє чітко розділити стабільну структуру файлової системи репозиторію та різноманітні операції над нею. Завдяки цьому можливі додавати нові функції, такі як збір статистики, очищення тимчасових файлів або експорт структури, створюючи нові класи-відвідувачі, не змінюючи при цьому вже написаний і протестований код класів файлів чи менеджерів. Це забезпечує дотримання принципу відкритості/закритості (Open/Closed Principle), роблячи архітектуру гнучкою до розширення без ризику зламати існуючий функціонал. Крім того, патерн допомагає уникнути засмічення класів бізнес-логіки зайвими методами, інкапсулюючи кожну специфічну операцію у власному класі. Нарешті, використання Visitor спрощує накопичення стану під час рекурсивного обходу дерева каталогів, дозволяючи зберігати проміжні результати (наприклад, розмір файлів) безпосередньо в об'єкті-відвідувачі.

Хід роботи

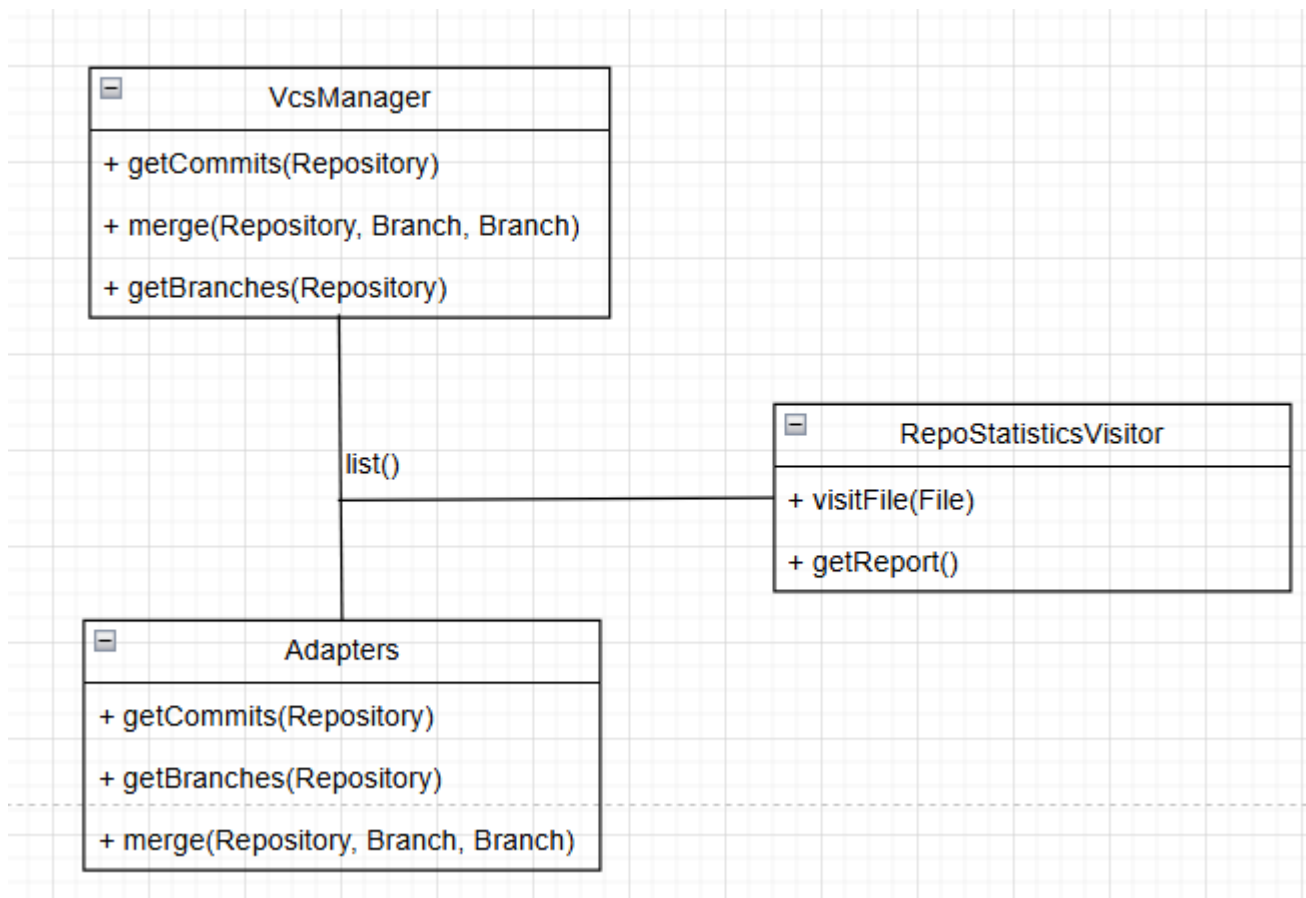


Рис 1 – Діаграма класів модулю з використанням паттерну «Відвідувач»

```
1 package vcs.repository.visitor;
2
3 import java.io.File;
4
5 public interface RepositoryVisitor {
6     void visitFile(File file);
7     void visitDirectory(File directory);
8 }
```

Рис 2 – інтерфейс RepositoryVisitor

```

1 package vcs.repository.visitor;
2
3 public interface Visitable { no usages 1 implem
4     void accept(RepositoryVisitor visitor);
5 }
6

```

Рис 3 – інтерфейс Visitor

```

1 package vcs.repository.visitor;
2
3 import java.io.File;
4
5 public class RepositoryContent implements Visitable { no usages
6     private final File rootDirectory; 3 usages
7
8     public RepositoryContent(String path) { no usages
9         this.rootDirectory = new File(path);
10    }
11
12    @Override no usages
13    public void accept(RepositoryVisitor visitor) {
14        if (rootDirectory.exists()) {
15            traverse(rootDirectory, visitor);
16        }
17    }
18
19    // Рекурсивний обхід дерева
20    @ private void traverse(File current, RepositoryVisitor visitor) { 2
21        // Ігноруємо папку самого VCS (.git, .svn, .hg)
22        if (current.getName().startsWith(".")) return;
23
24        if (current.isDirectory()) {
25            visitor.visitDirectory(current); // Відвідуємо папку
26
27            File[] files = current.listFiles();
28            if (files != null) {
29                for (File file : files) {
30                    traverse(file, visitor); // Йдемо вглиб
31                }
32            }
33        } else {
34            visitor.visitFile(current); // Відвідуємо файл
35        }
36    }
37 }
38

```

Рис 4 – клас RepositoryContent

```

1 package vcs.repository.visitor;
2
3 import java.io.File;
4
5 public class RepoStatisticsVisitor implements RepositoryVisitor { no usages
6     private long totalSize = 0; 2 usages
7     private int fileCount = 0; 2 usages
8     private int directoryCount = 0; 2 usages
9     private int javaFileCount = 0; 2 usages
10
11     @Override no usages
12     public void visitFile(File file) {
13         fileCount++;
14         totalSize += file.length();
15
16         if (file.getName().endsWith(".java")) {
17             javaFileCount++;
18         }
19     }
20
21     @Override no usages
22     public void visitDirectory(File directory) {
23         directoryCount++;
24     }
25
26     // Метод для отримання результату (звіту)
27     public String getReport() { no usages
28         return String.format(
29             "---- Repository Statistics ----\n" +
30             "Total Files: %d\n" +
31             "Directories: %d\n" +
32             "Java Source Files: %d\n" +
33             "Total Size: %.2f KB",
34             fileCount, directoryCount, javaFileCount, totalSize / 1024.0
35         );
36     }
37 }

```

Рис 5 – клас RepoStatisticsVisitor

Висновок

Отже, після виконання даної лабораторної роботи було детально вивчено поведінковий патерн проєктування «Відвідувач» (Visitor), досліджено його механізм подвійної диспетчеризації, сильні сторони у розділенні алгоритмів та структур даних, а також проаналізовано доцільність його використання. Для закріплення отриманих знань було реалізовано цей патерн у розроблюваному проєкті «VCS All-In-One». Було проведено аналіз функціональних потреб системи та виявлено необхідність виконання різномірних операцій над файловою

структурою репозиторію (таких як збір статистики, експорт або очищення) без внесення змін у класи самих файлів та папок. Імплементация механізму відвідувача дозволила винести цю логіку в окремі класи, забезпечивши дотримання принципу відкритості/закритості (Open/Closed Principle) та збереження чистоти коду основних сутностей проєкту.

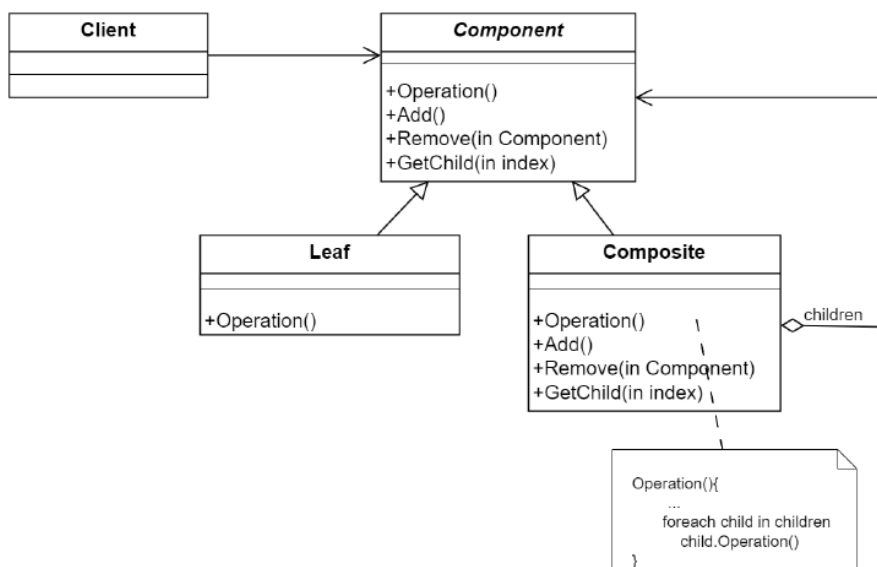
Відповіді на питання

1. Яке призначення шаблону «Композит»?

Призначення шаблону «Композит» (Composite) — об'єднати об'єкти в деревоподібні структури для представлення ієрархій «частина-ціле». Цей шаблон дозволяє клієнтам поводитися з окремими об'єктами та з композиціями (групами) об'єктів однаково.

Приклад: Файлова система, де папка може містити файли та інші папки. Команда "показати розмір" працює і для файлу, і для папки (яка сумує розмір свого вмісту).

2. Нарисуйте структуру шаблону «Композит».



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

- **Component (Компонент):** Абстракція (інтерфейс або абстрактний клас), що оголошує спільні операції як для простих, так і для складних об'єктів дерева.
- **Leaf (Лист):** Клас, що представляє кінцевий об'єкт (наприклад, файл). Він не має підлеглих елементів і виконує основну роботу.
- **Composite (Контейнер/Композит):** Клас, що містить колекцію дочірніх компонентів (як **Leaf**, так і інших **Composite**). Він не знає конкретних класів своїх дітей, працюючи з ними через інтерфейс **Component**.
- **Client (Клієнт):** Працює з усіма елементами через інтерфейс **Component**.

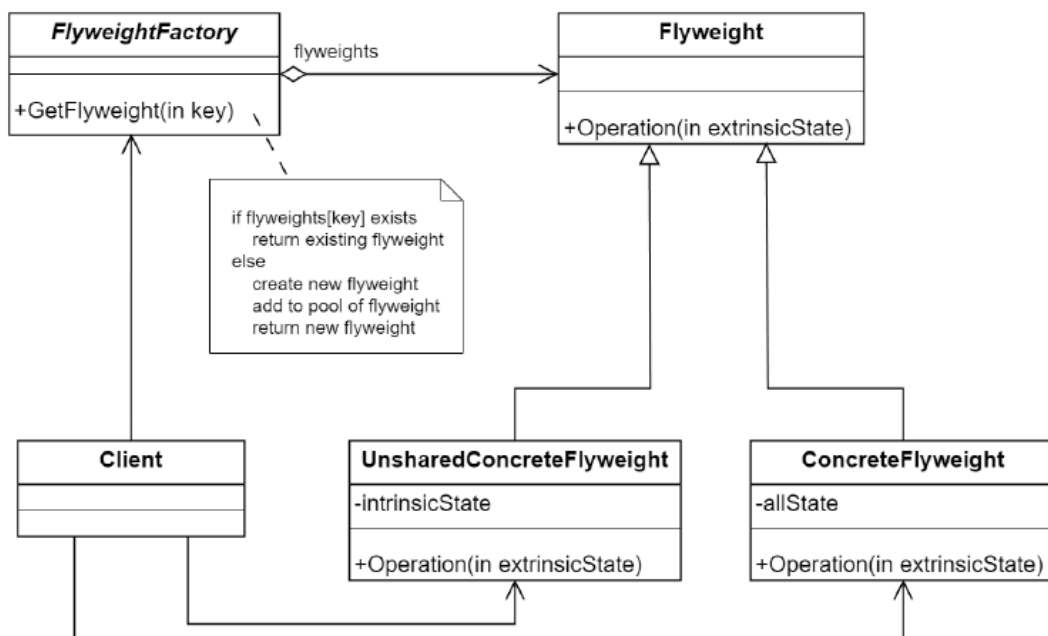
Взаємодія: Клієнт викликає метод `operation()` у об'єкта **Composite**. **Composite** проходить по своєму списку дітей і викликає `operation()` для кожного з них (делегування). Якщо дитина — це **Leaf**, вона виконує дію. Результати можуть агрегуватися і повертатися клієнту.

4. Яке призначення шаблону «Легковаговик»?

Призначення шаблону «Легковаговик» (Flyweight) — дозволити ефективно підтримувати величезну кількість дрібних об'єктів шляхом спільного використання (шарингу) тієї частини їхнього стану, яка є спільною для багатьох об'єктів (внутрішній стан), замість зберігання однакових даних у кожному об'єкті.

Приклад: Текстовий редактор. Замість зберігання зображення літери "А" в кожному об'єкті символу в документі, ми зберігаємо одне зображення "А" в пам'яті (легковаговик) і посилаємося на нього, додаючи лише координати (зовнішній стан).

5. Нарисуйте структуру шаблону «Легковаговик».



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

- **Flyweight (Легковаговик)**: Інтерфейс, через який клієнти спілкуються з легковаговиками. Приймає зовнішній стан (extrinsic state) як аргумент методу.
- **ConcreteFlyweight (Конкретний Легковаговик)**: Реалізує інтерфейс і зберігає внутрішній стан (intrinsic state), який є спільним і незмінним.
- **FlyweightFactory (Фабрика Легковаговиків)**: Створює і управляє пулом легковаговиків. Коли клієнт просить легковаговика, фабрика повертає існуючий екземпляр або створює новий, якщо такого ще немає.
- **Client (Клієнт)**: Зберігає зовнішній стан (унікальний для кожного об'єкта) і обчислює його перед викликом методу легковаговика.

Взаємодія: Клієнт звертається до Фабрики за об'єктом. Фабрика шукає його в кеші. Якщо знаходить — повертає посилання, якщо ні — створює. Клієнт викликає метод операції, передаючи унікальні дані (контекст) як параметри.

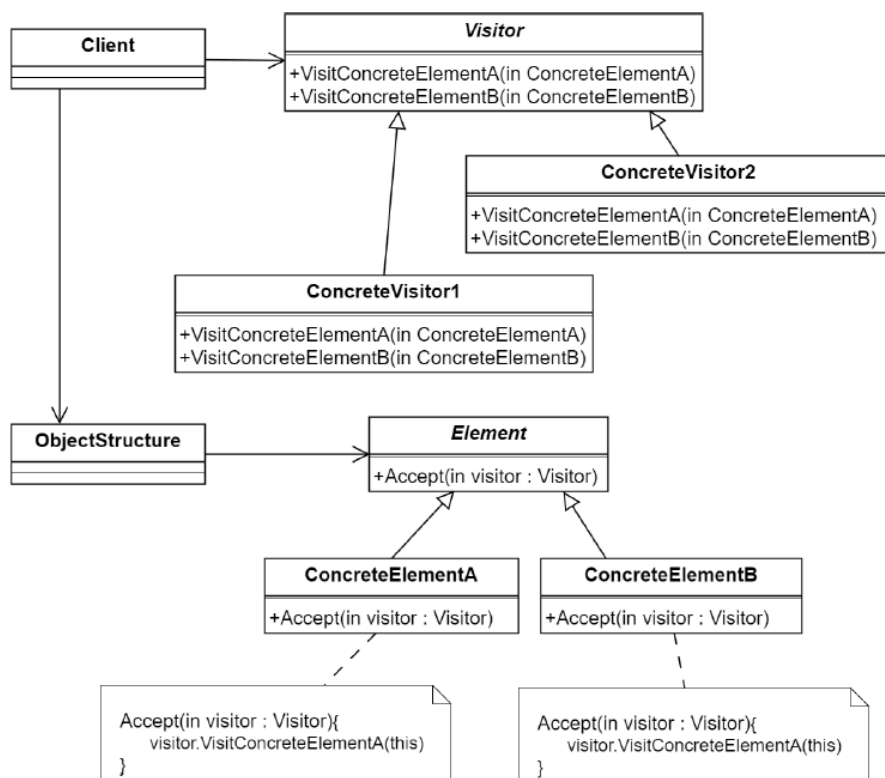
7. Яке призначення шаблону «Інтерпретатор»?

Призначення шаблону «Інтерпретатор» (Interpreter) — визначити граматику для певної мови та створити інтерпретатор, який використовує цю граматику для обробки (інтерпретації) речень цієї мови. Зазвичай використовується для часто повторюваних операцій, які можна описати як "речення" (наприклад, SQL-запити, математичні вирази, регулярні вирази).

8. Яке призначення шаблону «Відвідувач»?

Призначення шаблону «Відвідувач» (Visitor) — відокремити алгоритм від структури об'єктів, над якою він оперує. Це дозволяє додавати нові операції (функціонал) до ієрархії класів без зміни самих класів цих об'єктів.

9. Нарисуйте структуру шаблону «Відвідувач».



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

- **Visitor (Відвідувач):** Інтерфейс, що оголошує методи `visit` для кожного типу конкретного елемента (наприклад, `visitFile`, `visitDirectory`).

- **ConcreteVisitor** (Конкретний Відвідувач): Реалізує методи інтерфейсу **Visitor**, вкладаючи в них конкретну бізнес-логіку (наприклад, **XMLViewer**, **StatisticsCollector**).
- **Element** (Елемент): Інтерфейс, що оголошує метод `accept(Visitor v)`.
- **ConcreteElement** (Конкретний Елемент): Клас, що реалізує метод `accept`.

Взаємодія (Подвійна диспетчеризація):

1. Клієнт викликає `element.accept(visitor)`.
2. **ConcreteElement** всередині цього методу викликає `visitor.visitElement(this)`, передаючи себе як аргумент.
3. **Visitor** виконує потрібний метод для цього конкретного типу елемента, маючи доступ до його публічного інтерфейсу.