Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №8

з дисципліни «Теорія розробки програмного забезпечення»

Тема: «Патерни проектування»

Виконав:
Студент групи ІА-34
Цап Андрій

Перевірив:
Мягкий Михайло Юрійович

Київ – 2025

Зміст

Вступ

Мета: Вивчити структуру шаблонів «Composite», «Flyweight» (Пристосуванець), «Interpreter», «Visitor» та навчитися застосовувати їх в реалізації програмної системи.

У рамках лабораторної роботи було опрацьовано групу структурних та поведінкових шаблонів проєктування, серед яких Composite, Flyweight, Interpreter та Visitor. Після аналізу функціональних вимог системи керування версіями програмного забезпечення було визначено, що найбільш доцільним для інтеграції є шаблон **Composite**.

У системі версій є природна ієрархічна структура: кожна версія може містити підверсії, а підверсії — свої дочірні гілки. Цей механізм повністю відповідає моделі «дерева», де окремі елементи можуть бути як листками (кінцевими версіями), так і вузлами (версіями, що мають дочірні елементи).

Теоретичні відомості

Шаблон «Composite» - призначення: Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Даний шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю.

Переваги та недоліки:

+ Спрощує представлення деревоподібної структури.

+ Додає гнучкості в роботі з складними об’єктами та рекурсивними операціями.

+ Дозволяє додавати та видаляти об’єкти в ієрархії без впливу на клієнтський код.

- Потрібні додаткові зусилля для початкового впровадження.

- Вимагає гарно спроєктованого загального інтерфейсу.

Шаблон «Flyweight» - Призначення: Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.

Переваги та недоліки:

+ Заощаджує оперативну пам’ять.

- Витрачає процесорний час на пошук/обчислення контексту.

- Ускладнює код програми внаслідок введення безлічі додаткових класів.

Шаблон «Interpreter» - призначення: Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової). Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту.

Переваги та недоліки:

+ Граматику стає легко розширювати та змінювати, реалізації класів, що описують вузли абстрактного синтаксичного дерева схожі (легко кодуються).

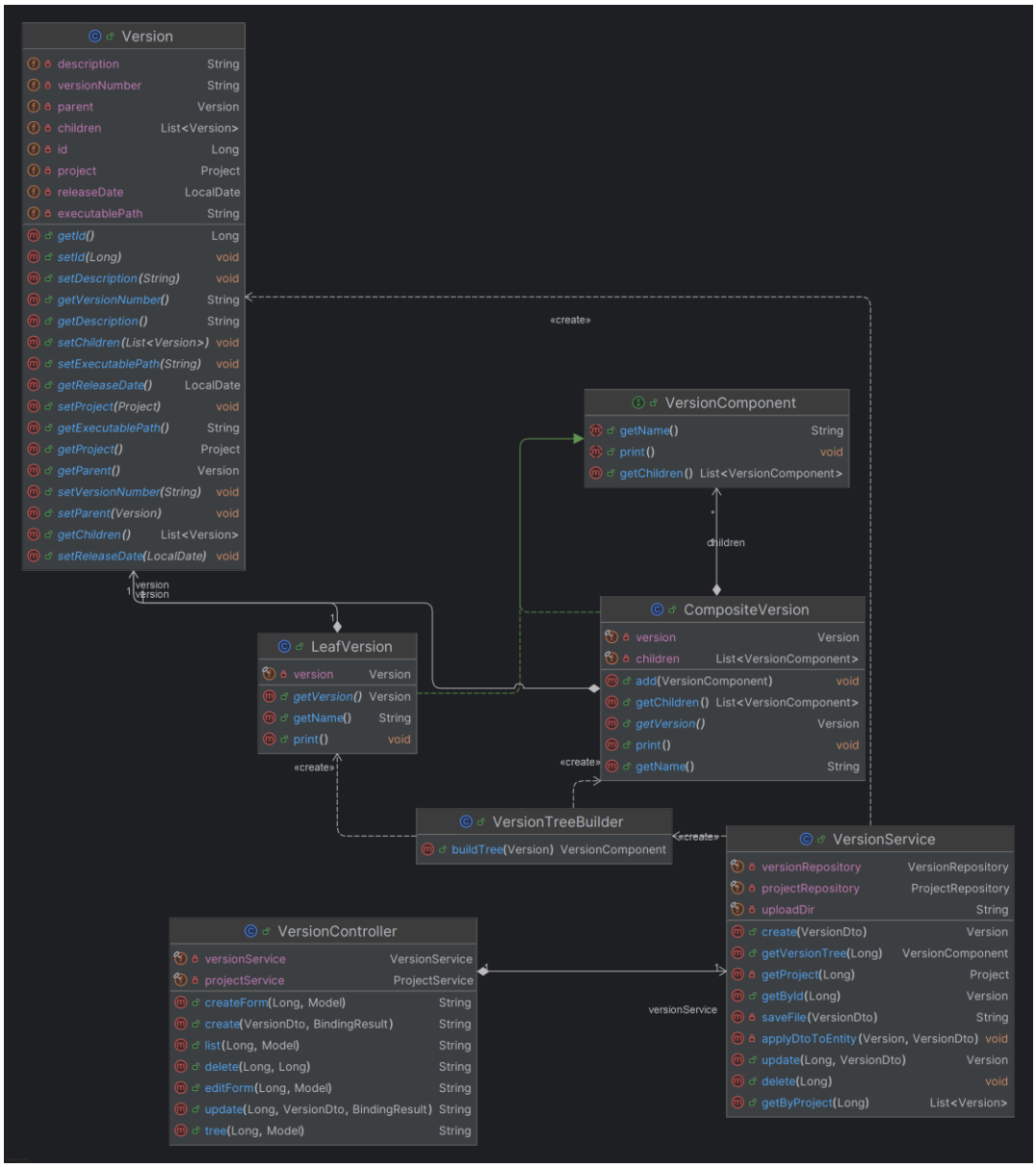
+ Можна легко змінювати спосіб обчислення виразів.

- Супроводження граматики с великою кількістю правил є проблематичним.

Шаблон «Visitor» - призначення: Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів. 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.

Хід роботи

Діаграма класів для реалізації шаблону Composite:



VersionComponent – базовий інтерфейс шаблону Composite.

Визначає спільні операції для всіх елементів дерева версій:

- getName() – повертає назву версії;
- getChildren() – повертає список дочірніх компонентів (за замовчуванням порожній).

Інтерфейс задає єдиний контракт для листків та складених об’єктів.

LeafVersion – клас, що реалізує Leaf у структурі Composite.
Представляє кінцеву версію, яка не має дочірніх елементів. Містить:

- поле version (об’єкт доменної моделі Version);
- реалізацію getName() через номер версії;

CompositeVersion – клас складеного елемента (Composite).
Може містити інші компоненти, що утворюють дерево версій. Містить:

- поле version, що описує поточну версію;
- список children типу List<VersionComponent>;
- метод add(), що додає дочірні компоненти;
- реалізації getChildren(), getName(), які працюють рекурсивно та проходять по всій ієрархії.

CompositeVersion дозволяє поводитися з групою версій так само, як з одним об’єктом.

VersionTreeBuilder – клас-утиліта, який створює дерево версій.
Містить метод buildTree(Version), що:

- перевіряє, чи має версія дочірні елементи;
- якщо ні – створює LeafVersion;
- якщо так – створює CompositeVersion і рекурсивно додає підверсії до його структури.
  Виступає механізмом побудови ієрархії відповідно до структури доменної моделі Version.

Version – доменна модель, на основі якої формується дерево Composite.
Містить інформацію про версію програмного забезпечення: номер, опис, дату релізу, посилання на батьківську версію та список дочірніх версій.

VersionService – сервісний компонент, який працює з доменною моделлю Version.
Забезпечує операції отримання версій, редагування, видалення та побудови дерева.

VersionController – контролер, який забезпечує роботу з версіями через веб-інтерфейс.

Переваги використання

Об’єднана робота з усіма об’єктами

Не потрібно писати окремі перевірки - обидва варіанти мають однаковий інтерфейс.

Легке розширення функціоналу

Якщо потрібно додати нову поведінку - достатньо додати новий метод у VersionComponent та реалізувати його у LeafVersion і CompositeVersion.
Не потрібно змінювати структуру або переписувати лінійний код.

Недоліки використання

Ускладнення структури та збільшення кількості класів

Для реалізації ієрархії доводиться створювати щонайменше три окремих компоненти:
VersionComponent, CompositeVersion, LeafVersion.
Структура стає менш очевидною, ніж проста робота зі списком.

Рекурсивний обхід не такий легкий

Обхід дерева через getChildren() відбувається рекурсивно.

Новачку складніше зрозуміти, в якому порядку викликаються методи й як формується фінальний результат.

Вихідні коди реалізації шаблону Composite

VersionComponent.java

```java
package org.example.projectmanagementsoftware.pattern.composite.interfaces;

import java.util.List;

public interface VersionComponent {  14 usages  2 implementations  new *
    String getName();  2 implementations  new *
    default List<VersionComponent> getChildren() {  no usages  1 override  new *
        return List.of();
    }
}
```

Рис 1 - Код VersionComponent

LeafVersion.java

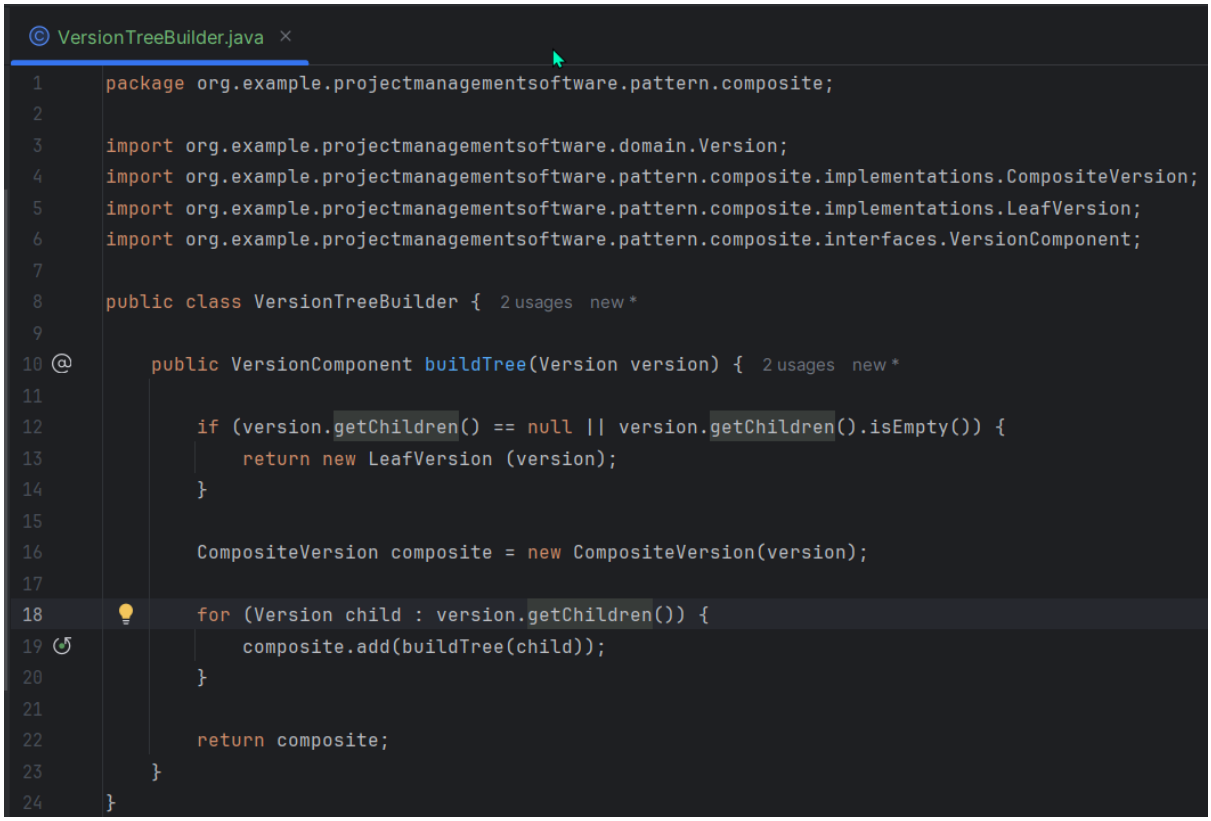
```java
package org.example.projectmanagementsoftware.pattern.composite.implementations;

import lombok.Getter;
import lombok.Setter;
import org.example.projectmanagementsoftware.domain.Version;
import org.example.projectmanagementsoftware.pattern.composite.interfaces.VersionComponent;

@Setter  2 usages  new *
@Getter
public class LeafVersion implements VersionComponent {

    private final Version version;

    public LeafVersion(Version version) {  1 usage  new *
        this.version = version;
    }

    @Override  new *
    public String getName() {
        return version.getVersionNumber();
    }

}
```

Рис 2 - Код LeafVersion

CompositeVersion.java

```java
package org.example.projectmanagementsoftware.pattern.composite.implementations;

import lombok.Getter;
import org.example.projectmanagementsoftware.domain.Version;
import org.example.projectmanagementsoftware.pattern.composite.interfaces.VersionComponent;

import java.util.ArrayList;
import java.util.List;

@Getter  3 usages  new *
public class CompositeVersion implements VersionComponent {

    private final Version version;
    private final List<VersionComponent> children = new ArrayList<> ();  2 usages

    public CompositeVersion(Version version) {  1 usage  new *
        this.version = version;
    }

    public void add(VersionComponent component) {  new *
        children.add(component);
    }

    @Override  new *
    public String getName() {
        return version.getVersionNumber();
    }

    @Override  no usages  new *
    public List<VersionComponent> getChildren() {
        return children;
    }
}
```

Рис 3 - Код CompositeVersion



```java
package org.example.projectmanagementsoftware.pattern.composite;

import org.example.projectmanagementsoftware.domain.Version;
import org.example.projectmanagementsoftware.pattern.composite.implementations.CompositeVersion;
import org.example.projectmanagementsoftware.pattern.composite.implementations.LeafVersion;
import org.example.projectmanagementsoftware.pattern.composite.interfaces.VersionComponent;

public class VersionTreeBuilder {  2 usages  new *

    public VersionComponent buildTree(Version version) {  2 usages  new *

        if (version.getChildren() == null || version.getChildren().isEmpty()) {
            return new LeafVersion (version);
        }

        CompositeVersion composite = new CompositeVersion(version);

        for (Version child : version.getChildren()) {
            composite.add(buildTree(child));
        }

        return composite;
    }
}
```

Рис 4 - Код VersionBuilder

```java
public VersionComponent getVersionTree(Long rootId) {  1 usage  new *
    return new VersionTreeBuilder().buildTree(getById(rootId));
}
```

Рис 5 - Новий метод в класі VersionService

```java
@GetMapping("/{id}/tree")  new *
public String tree(@PathVariable Long id, Model model) {
    VersionComponent tree = versionService.getVersionTree(id);
    model.addAttribute( attributeName: "tree", tree);
    return "versions/tree";
}
```

Рис 6 - Новий метод в класі VersionController

Висновки

Під час виконання лабораторної роботи були опрацьовані шаблони Composite, Flyweight, Interpreter та Visitor. Для кожного шаблону було розглянуто принципи роботи, особливості застосування та переваги у реальних програмних системах. На основі аналізу архітектури застосунку було визначено, що найкраще підходить саме шаблон **Composite**, оскільки він дозволяє природно відобразити ієрархію версій, забезпечити зручну організацію дерева та уніфікувати роботу з різними типами компонентів.

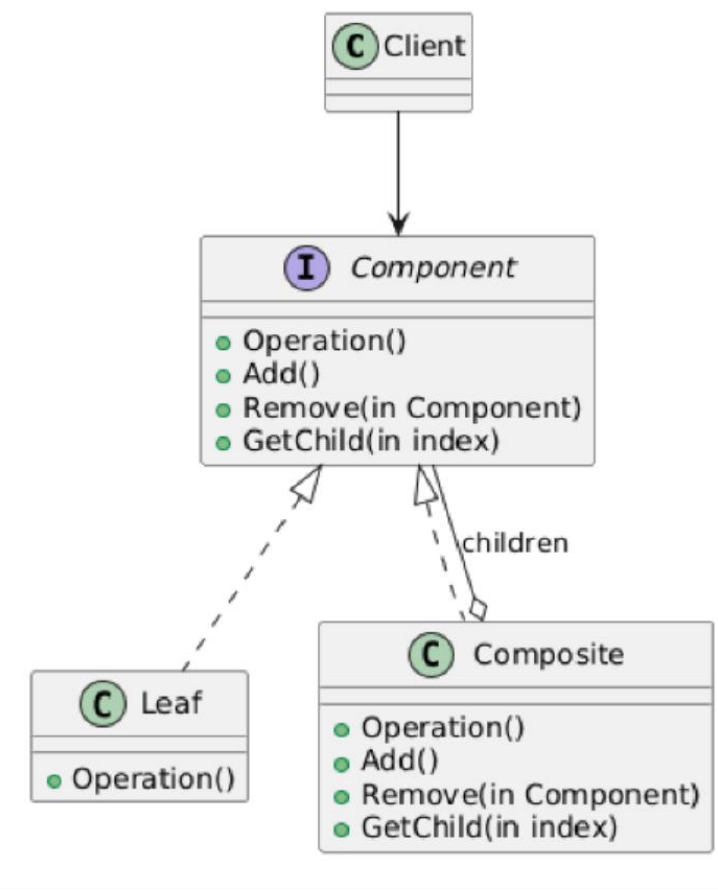
Застосування Composite у модулі керування версіями дало можливість створити гнучку, масштабовану структуру, що легко розширюється додаванням нових типів вузлів. Такий підхід зменшує зв’язність між класами, спрощує підтримку системи та забезпечує чисту архітектуру, придатну до подальшого розвитку.

Контрольні запитання

1. Яке призначення шаблону «Композит»?

Шаблон призначений для представлення ієрархічних структур типу «дерево», де окремі об’єкти та їхні групи обробляються однаково. Дозволяє клієнту працювати з Leaf і Composite через спільний інтерфейс.

2. Нарисуйте структуру шаблону «Композит».



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

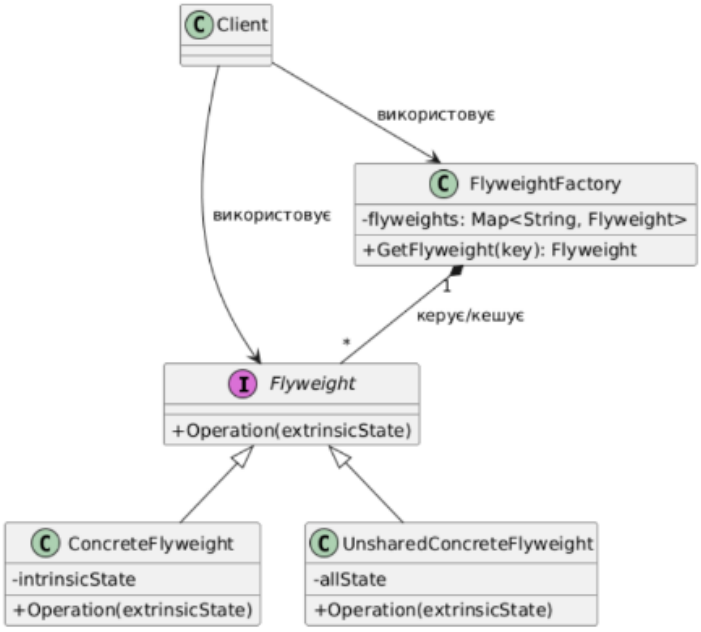
- Component — базовий інтерфейс з методами Operation(), Add(), Remove(), GetChild().
- Leaf — кінцевий елемент, реалізує лише Operation().
- Composite — вузол, містить колекцію Component і виконує Operation() рекурсивно для всіх дітей.

Взаємодія: клієнт працює через Component; Composite включає Leaf або інші Composite; виклики Operation() проходять по дереву рекурсивно.

4. Яке призначення шаблону «Легковаговик»?

Шаблон призначений для зменшення кількості об’єктів шляхом спільного використання внутрішнього стану. Внутрішній стан зберігається у Flyweight-об’єкті, а зовнішній надається клієнтом під час використання.

5. Нарисуйте структуру шаблону «Легковаговик».



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

- Flyweight — інтерфейс з методом Operation(extrinsicState).
- ConcreteFlyweight — реалізація, зберігає внутрішній стан.
- FlyweightFactory — створює й кешує Flyweight-об’єкти.
- Client — передає зовнішні дані та викликає Operation().

Взаємодія: клієнт отримує Flyweight з фабрики; фабрика повертає існуючий об’єкт або створює новий; зовнішній стан передається під час виклику Operation().

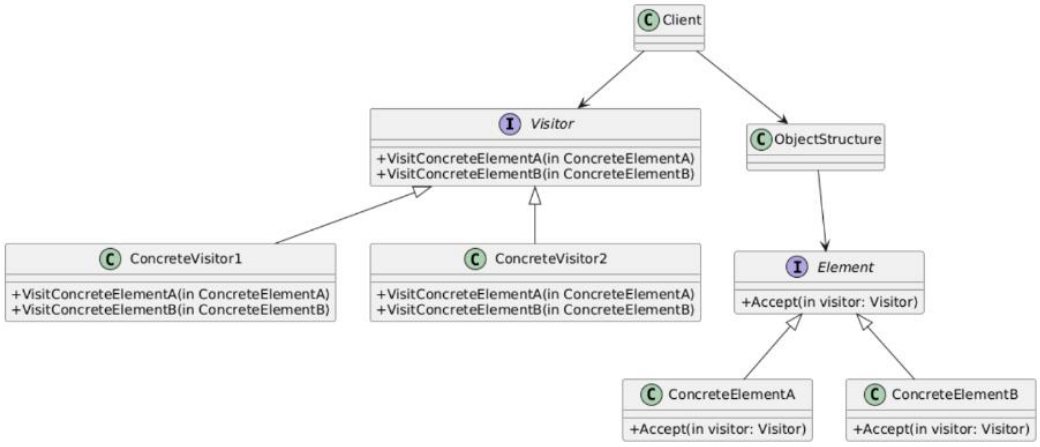
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон використовується для опису граматики та побудови інтерпретатора, який може виконувати вирази певної мови (простих DSL, команд, правил). Кожен клас представляє елемент граматики.

8. Яке призначення шаблону «Відвідувач»?

Шаблон дозволяє додавати нові операції до об’єктів складних структур без зміни самих класів елементів. Розділяє структуру об’єктів і операції над ними.

9. Нарисуйте структуру шаблону «Відвідувач».



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

- Visitor — містить набір visit-методів для різних типів елементів.
- ConcreteVisitor — реалізує конкретні операції над елементами.
- Element — базовий інтерфейс з методом Accept(Visitor).
- ConcreteElement — передає себе у Visitor через accept.
- ObjectStructure — зберігає набір елементів і дозволяє обійти їх.

Взаємодія: клієнт створює Visitor; кожен елемент викликає visitor.visit(this); Visitor виконує потрібну операцію відповідно до типу елемента.