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: «Особиста бухгалтерія»

Виконала:
Студентка групи ІА-34
Дригант А.С
Перевірив:
Мягкий Михайло Юрійович

Київ-2025

Тема: Патерни проектування

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

Тема роботи:

27. Особиста бухгалтерія (state, prototype, decorator, bridge, flyweight, SOA) Програма повинна бути наочним засобом для ведення особистих фінансів: витрат і прибутку; з можливістю встановлення періодичних витрат / прибутку (зарплата і орендна плата); введення сканованих чеків з відповідними статтями витрат; побудова статистики; експорт/імпорт в Excel, реляційні джерела даних; різні рахунки; ведення єдиного фонду на всі рахунки (всією сім'єю) – на особливі потреби (ремонт, автомобіль, відпустка); можливість введення вкладів / кредитів для контролю банківських рахунків (звірка нарахованих відсотків з необхідними і т.д.).

Теоритичні відомості

Шаблон «Composite» (Компонувальник)

Призначення:

Дозволяє об’єднувати об’єкти в деревоподібну структуру для представлення ієрархій типу «частина-ціле». Дає змогу працювати з окремими елементами й групами елементів однаково.

Суть:

Використовується, коли потрібно обробляти ієрархії об’єктів.

Наприклад, у формі є дочірні елементи (поля, кнопки, написи), які теж можуть містити інші елементи.

Переваги:

- Спрощує роботу з деревоподібними структурами.
- Дає гнучкість у додаванні/видаленні елементів.

- Полегшує рекурсивні операції.

Недоліки:

- Потребує ретельного проєктування спільного інтерфейсу.
- Складніше впровадження на початку.

Шаблон «Flyweight»

Призначення:

Зменшує кількість об’єктів, розділяючи спільні дані між ними.

Суть:

Має внутрішній стан (спільний для всіх) і зовнішній стан (індивідуальний для контексту використання).

Наприклад, усі однакові літери тексту представлені одним об’єктом, який багаторазово використовується.

Переваги:

- Економить оперативну пам’ять.

Недоліки:

- Потрібен додатковий процесорний час на пошук і відновлення контексту.
- Код стає складнішим через додаткові класи.

Шаблон «Interpreter» (Інтерпретатор)

Призначення:

Використовується для створення граматики простої мови та її інтерпретатора.

Суть:

Формує абстрактне синтаксичне дерево, де кожен вузол — це правило або операція. Кожен вираз інтерпретується з урахуванням контексту.

Підходить для невеликих і простих мов або задач із часто змінюваними правилами.

Переваги:

- Легко змінювати і розширювати граматику.
- Просто додавати нові способи обчислення.

Недоліки:

При великій кількості правил код стає громіздким.

Шаблон «Visitor» (Відвідувач)

Призначення:

Дозволяє визначати операції над елементами об’єктної структури без зміни класів цих елементів.

Суть:

Виносить логіку дій (наприклад, розрахунки) в окремий клас “відвідувача”.

Елементи приймають відвідувача і викликають у нього метод, який відповідає їхньому типу.

Зручно для реалізації різних обчислень над тими ж об’єктами.

Переваги:

- Легко додавати нові операції (нові відвідувачі).
- Дані і логіка розділені.

Недоліки:

- Важко додавати нові типи елементів (бо потрібно змінювати всіх відвідувачів).

Хід роботи

Діаграма класів

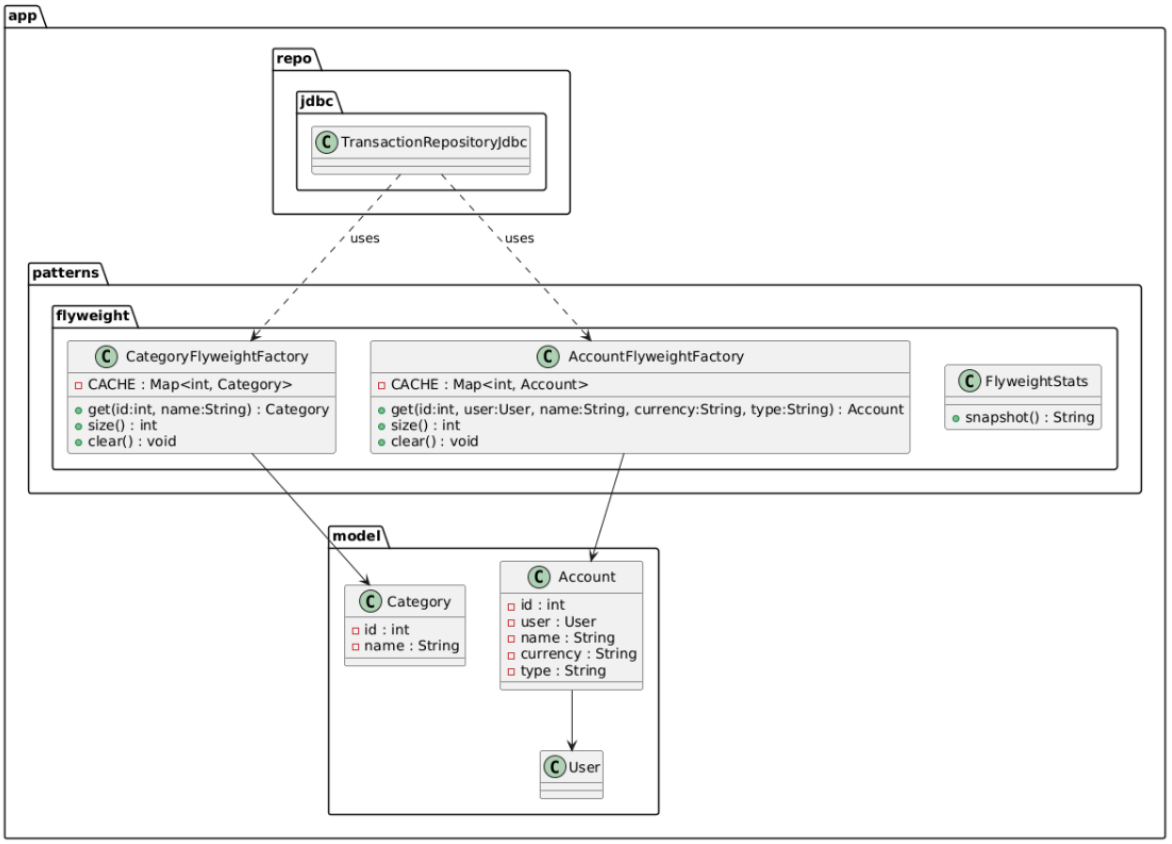


Рисунок - 8.1 - Діаграма класів реалізації шаблону «Flyweight»

На діаграмі зображено реалізацію шаблону Flyweight (Пристосуванець) у системі фінансового обліку. Його мета - зменшити кількість створюваних об’єктів під час завантаження даних із бази, повторно використовуючи однакові екземпляри Account і Category. У пакеті app.model містяться класи предметної області: Account, який описує рахунок користувача та має внутрішній спільний стан (id, user, name, currency, type), і Category, що представляє категорію транзакції з полями id та name. У пакеті app.patterns.flyweight розміщені фабрики AccountFlyweightFactory та CategoryFlyweightFactory, які керують кешем об’єктів і повертають уже існуючі екземпляри замість створення нових, а також допоміжний клас FlyweightStats, який показує кількість об’єктів у кеші. У пакеті

app.repo.jdbc клас TransactionRepositoryJdbc відповідає за вибірку транзакцій із бази даних і використовує обидві фабрики, щоб при мапінгу ResultSet отримувати об’єкти Account і Category через кеш, а не створювати їх напряму. Таким чином, шаблон Flyweight дозволяє оптимізувати використання пам’яті, зменшуючи дублювання даних і підвищуючи ефективність роботи системи

1) AccountFlyweightFactory

```java
import app.model.Account;
import app.model.User;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

1 usage  new *
public final class AccountFlyweightFactory {
    3 usages
    private static final Map<Integer, Account> CACHE = new ConcurrentHashMap<>();

    no usages  new *
    private AccountFlyweightFactory() {}

    new *
    public static Account get(int id, User user, String name, String currency, String type) {
        return CACHE.compute(id, (k, existing) -> {
            if (existing == null) {
                Account a = new Account();
                a.setId(id);
                a.setUser(user);
                a.setName(name);
                a.setCurrency(currency);
                a.setType(type);
                return a;
            } else {
                if (existing.getUser() == null && user != null) existing.setUser(user);
                if (existing.getName() == null && name != null) existing.setName(name);
                if (existing.getCurrency() == null && currency != null) existing.setCurrency(currency);
                if (existing.getType() == null && type != null) existing.setType(type);
                return existing;
            }
        });
    }

    new *
    public static int size() { return CACHE.size(); }
    no usages  new *
    public static void clear() { CACHE.clear(); }
}
```

Рисунок - 8.2 - Клас AccountFlyweightFactory

Клас AccountFlyweightFactory відповідає за реалізацію шаблону Flyweight (Пристосуванець) для об’єктів типу Account, які описують рахунки користувачів.

Основне його завдання - зберігати створені екземпляри у кеші ConcurrentHashMap, щоб однакові рахунки з однаковим id не створювалися повторно. Метод get() є головним у фабриці: він перевіряє, чи існує об’єкт у кеші, і якщо ні - створює новий Account, встановлюючи основні властивості (user, name, currency, type), а якщо вже є - повертає існуючий, при цьому може доповнювати відсутні дані. Таке рішення дозволяє значно зменшити кількість створюваних об’єктів у пам’яті, забезпечує узгодженість даних між різними частинами програми та підвищує продуктивність при роботі з великою кількістю транзакцій. Додаткові методи size() і clear() дають змогу перевірити кількість збережених об’єктів і очищати кеш у разі потреби.

2) CategoryFlyweightFactory

```java
package app.patterns.flyweight;

import app.model.Category;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

1 usage  new *
public final class CategoryFlyweightFactory {
    3 usages
    private static final Map<Integer, Category> CACHE = new ConcurrentHashMap<>();

    no usages  new *
    private CategoryFlyweightFactory() {}

    new *
    public static Category get(int id, String name  ) {
        return CACHE.compute(id, (k, existing) -> {
            if (existing == null) {
                Category c = new Category();
                c.setId(id);
                if (name != null) c.setName(name);
                return c;
            } else {

                if (existing.getName() == null && name != null) {
                    existing.setName(name);
                }
                return existing;
            }
        });
    }
    new *
    public static int size() { return CACHE.size(); }
    no usages  new *
    public static void clear() { CACHE.clear(); }
}
```

Рисунок - 8.3 - Клас CategoryFlyweightFactory

Клас CategoryFlyweightFactory реалізує шаблон Flyweight (Пристосуванець) для об’єктів Category, які представляють категорії транзакцій. Його основна мета - зберігати створені екземпляри категорій у спільному кеші, щоб уникнути дублювання однакових об’єктів у пам’яті. Для цього використовується потокобезпечна колекція ConcurrentHashMap, де ключем є ідентифікатор категорії id, а значенням - об’єкт Category. Метод get() перевіряє наявність екземпляра у кеші: якщо його немає, створює новий об’єкт і додає в колекцію; якщо вже існує - повертає раніше створений, оновлюючи поле name, якщо воно порожнє. Таке рішення дозволяє ефективно використовувати пам’ять і забезпечує узгодженість даних при повторному зверненні до одних і тих самих категорій. Допоміжні методи size() та clear() використовуються для контролю кількості об’єктів у кеші та його очищення за потреби.

3. FlyweightStats

```
package app.patterns.flyweight;

no usages  new *
public final class FlyweightStats {
    no usages  new *
    private FlyweightStats() {}
    no usages  new *
    public static String snapshot() {
        return "Flyweights -> accounts: " + AccountFlyweightFactory.size()
                + ", categories: " + CategoryFlyweightFactory.size();
    }
}
```

Рисунок 8.4 - Клас FlyweightStats

Клас FlyweightStats є допоміжним утилітним класом, який використовується для моніторингу роботи шаблону Flyweight у програмі. Його головна мета - відображати поточну кількість створених і кешованих об’єктів у фабриках AccountFlyweightFactory та CategoryFlyweightFactory. Клас має лише один публічний статичний метод snapshot()

4) Використання у TransactionRepositoryJdbc

```java
package app.repo;

import app.model.User;
import app.model.Transaction;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.List;
import java.util.Map;

1 implementation  Nastenaaa *
public interface TransactionRepository {
    1 implementation  Nastenaaa
    Transaction save(Transaction t);
    4 usages  1 implementation  Nastenaaa
    List<Transaction> findByUserAndPeriod(User user, LocalDate from, LocalDate to);
    2 usages  1 implementation  Nastenaaa
    Map<Integer, BigDecimal> balancesByUser(int userId);
    2 usages  1 implementation  Nastenaaa
    BigDecimal balanceForAccount(int accountId);
}
```

Рисунок 8.5 - Використання шаблону Flyweight у класі TransactionRepositoryJdbc

На рисунку показано, як шаблон Flyweight (Пристосуванець) застосовується в репозиторії TransactionRepositoryJdbc під час отримання даних із бази. У процесі мапінгу результатів запиту ResultSet на об’єкти Transaction, для створення об’єктів Account і Category використовуються фабрики AccountFlyweightFactory та CategoryFlyweightFactory. Це означає, що замість створення нових екземплярів при кожному рядку результату, система перевіряє наявність об’єкта з таким самим id у кеші й повертає вже існуючий. Таким чином, у пам’яті зберігається лише один спільний об’єкт для кожного унікального рахунку та категорії, що значно знижує дублювання даних і підвищує ефективність роботи системи. Шаблон Flyweight тут забезпечує оптимізацію ресурсів і стабільність при роботі з великими обсягами транзакцій.

5) Використання в UI

```
for (int i = 0; i < current.size(); i++) {
    var t = current.get(i);


    if (t.getAccount() != null) {
        var a0 = t.getAccount();
        var a  = app.patterns.flyweight.AccountFlyweightFactory.get(
                a0.getId(),
                a0.getUser() != null ? a0.getUser() : user,
                a0.getName(),
                a0.getCurrency(),
                a0.getType()
        );
        t.setAccount(a);
    }


    if (t.getCategory() != null) {
        var c0 = t.getCategory();
        var c  = app.patterns.flyweight.CategoryFlyweightFactory.get(
                c0.getId(),
                c0.getName()
        );
        t.setCategory(c);
    }
}
```

Рисунок 8.6 - Використання шаблону Flyweight у класі TransactionsPanel

На рисунку показано, як у графічному інтерфейсі реалізовано використання шаблону Flyweight (Пристосуванець) для оптимізації роботи з об’єктами Account та Category. Після завантаження транзакцій із бази даних усі отримані об’єкти нормалізуються через фабрики AccountFlyweightFactory та CategoryFlyweightFactory. Це означає, що для кожного унікального id створюється лише один спільний екземпляр рахунку або категорії, який потім повторно використовується у списку транзакцій. Такий підхід дозволяє зменшити кількість об’єктів у пам’яті, уникнути дублювання та підвищити ефективність відображення даних у таблиці. Таким чином, у графічному інтерфейсі шаблон Flyweight забезпечує оптимальну роботу програми навіть при великій кількості транзакцій.

Результат:

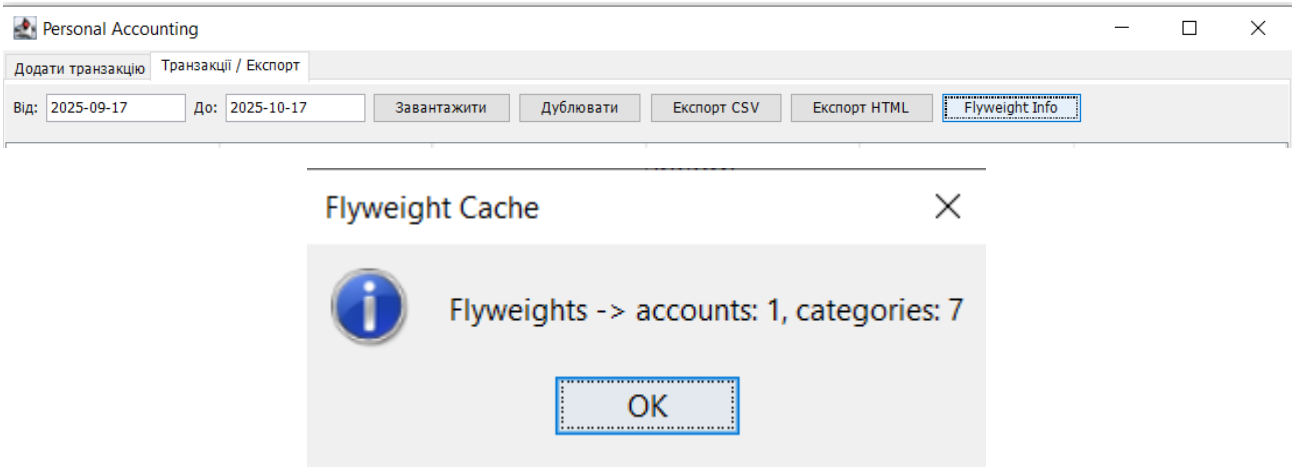


Рисунок 8.7 - Результат роботи шаблону Flyweight у застосунку

На рисунку продемонстровано результат виклику методу FlyweightStats.snapshot(), який показує поточний стан кешу Flyweight.
Це означає, що у системі створено лише один об’єкт Account та сім об’єктів Category, незважаючи на те, що транзакцій може бути значно більше. Такий результат підтверджує правильну роботу шаблону Flyweight (Пристосуванець): об’єкти з однаковими даними не дублюються, а повторно використовуються з кешу. Завдяки цьому зменшується кількість створюваних екземплярів у пам’яті, підвищується продуктивність і ефективність використання ресурсів програми.

Висновок

У цій лабораторній роботі я реалізувала шаблон проектування Flyweight (Пристосуванець) для оптимізації використання пам’яті в системі фінансового обліку. Під час завантаження транзакцій об’єкти Account і Category створюються не напряму, а через фабрики AccountFlyweightFactory та CategoryFlyweightFactory, які кешують уже існуючі екземпляри. Це дало змогу уникнути дублювання однакових даних у пам’яті й забезпечити повторне використання спільних об’єктів.

У графічному інтерфейсі я додала кнопку Flyweight Info, яка показує поточну кількість збережених об’єктів у кеші. Після виконання програми отримано результат accounts: 1, categories: 7, що підтвердило ефективну роботу

шаблону та правильність його інтеграції.

Завдяки застосуванню Flyweight система стала більш продуктивною, економною щодо ресурсів і масштабованою. Реалізація цього шаблону показала, як можна ефективно управляти пам’яттю та уникати надлишкового створення об’єктів у програмі.

Контрольні питання

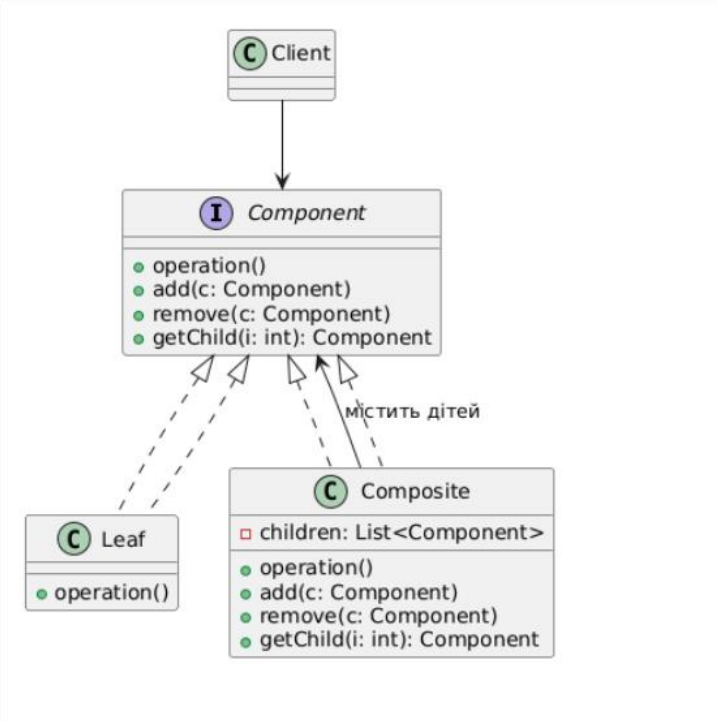
1. Яке призначення шаблону «Композит»?

Шаблон «Композит» (Composite) використовується для представлення ієрархічних структур типу «ціле–частина».

Він дозволяє об’єднувати об’єкти у деревоподібні структури та працювати з ними однаково, незалежно від того, чи це окремий об’єкт, чи група об’єктів.

Тобто клієнт може викликати однакові методи для окремого елемента або для контейнера, що містить інші елементи.

2. Нарисуйте структуру шаблону «Композит».



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

- Component спільний інтерфейс для листків і вузлів; клієнт працює тільки з ним.

- Leaf- кінцевий елемент без дітей; реалізує реальну роботу в operation().
- Composite - контейнер з колекцією Component; у operation() ітерується по дітях і делегує виклик.
- Client - викликає методи через Component, не розрізняючи тип (листок/вузол).

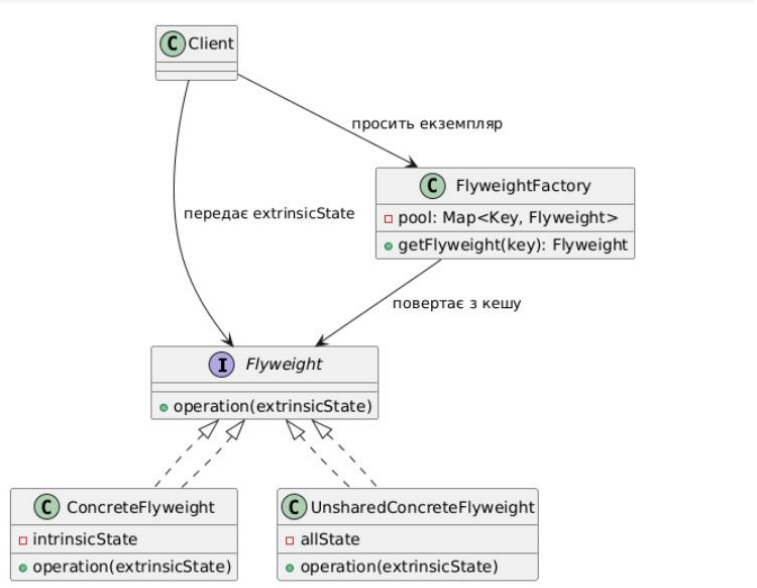
4. Яке призначення шаблону «Легковаговик»?

Оптимізація пам’яті при великій кількості дрібних однотипних об’єктів. Патерн розділяє стан на:

- intrinsic (внутрішній, незмінний і спільний, зберігається в об’єкті Flyweight),
- extrinsic (зовнішній, змінний, передається клієнтом під час виклику).

Замість тисяч дубльованих екземплярів тримаємо невеликий пул спільних об’єктів.

5. Нарисуйте структуру шаблону «Легковаговик».



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

- Flyweight - інтерфейс з операцією, що приймає extrinsic стан.
- ConcreteFlyweight - містить intrinsic стан; реалізує логіку в operation().
- UnsharedConcreteFlyweight - варіант без спільного використання (коли екземпляр унікальний).
- FlyweightFactory - керує пулом об’єктів (кеш), повертає вже створені

екземпляри за ключем.

- Client - не зберігає внутрішній стан у flyweight; при кожному виклику подає extrinsic.

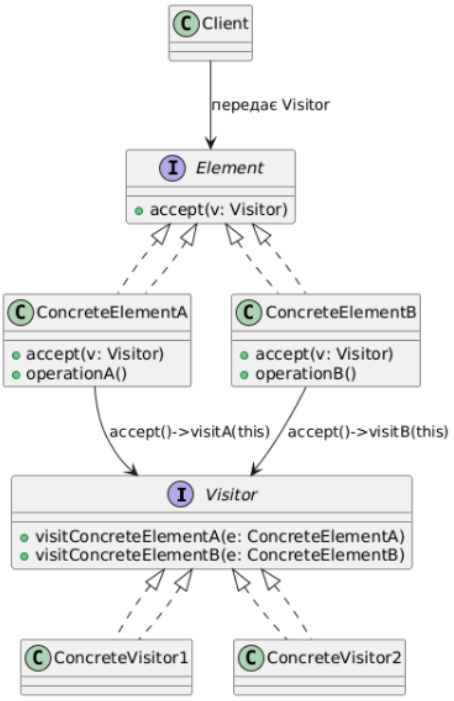
7. Яке призначення шаблону «Інтерпретатор»?

Надає спосіб визначити просту граматику предметної мови і інтерпретувати вирази цієї мови через набір класів-правил. Корисно, коли є багато повторюваних виразів/правил і потрібна гнучка побудова/обчислення (логічні вирази, арифметичні формули, фільтри).

8. Яке призначення шаблону «Відвідувач»?

Дозволяє додавати нові операції до сімейства класів без зміни їхньої структури. Елементи приймають відвідувача (accept(visitor)), а той виконує специфічну дію, маючи перевантажені visitXxx(...) для різних типів елементів. Зручно для операцій на складних структурах (напр., обхід дерева з різними «обробками»).

9. Нарисуйте структуру шаблону «Відвідувач».



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

Element - інтерфейс елементів зі способом accept(Visitor).

ConcreteElementA/B - конкретні елементи; в accept() викликають відповідний visitXxx(this).

Visitor - інтерфейс відвідувача з методами visit... для кожного типу елементів.

ConcreteVisitor1/2 - реалізації різних операцій (аналіз, підрахунок, експортування тощо).

Client - формує набір елементів і «пропускає» через них потрібного відвідувача.