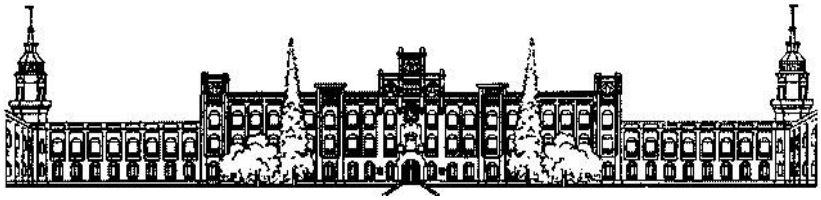


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №8

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав:

студент групи ІА-32

Нагорний Максим

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

Вступ

Тема: Вступ до паттернів проектування.

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

Завдання

- Ознайомитись з короткими теоретичними відомостями.

- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

- Реалізувати один з розглянутих шаблонів за обраною темою.

- Реалізувати не менше 3-х класів відповідно до обраної теми.

- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Composite»

Шаблон «Composite» належить до структурних шаблонів проєктування та використовується для організації об’єктів у вигляді деревоподібних структур, де окремі елементи та їхні групи повинні оброблятися однаково. Основна ідея полягає в тому, щоб надати клієнтському коду можливість працювати з одиничними та складеними об’єктами через єдиний інтерфейс, не розрізняючи їхнього конкретного типу. У шаблоні кожен елемент дерева реалізує спільний контракт, який визначає базові операції. Листовий елемент не містить дочірніх компонентів, тоді як складений елемент-композит зберігає колекцію дочірніх об’єктів і делегує їм виконання операцій. Такий підхід дозволяє будувати складні ієрархії, зручні для обходу та модифікації. Composite спрощує розширюваність системи та роботу з вкладеними структурами, оскільки клієнт не зобов’язаний знати, чи працює він з простим компонентом або з контейнером, що містить інші елементи. Це робить його корисним у графічних інтерфейсах, файлових системах, ієрархіях організацій та будь-яких структурах, де елементи складаються з інших елементів.

Шаблон «Flyweight»

Шаблон «Flyweight» є структурним шаблоном, що призначений для економії пам’яті у випадках, коли система використовує велику кількість об’єктів з однаковим внутрішнім станом. Основною ідеєю є поділ стану об’єкта на внутрішній та зовнішній. Внутрішній стан є незмінним і може бути спільним для багатьох екземплярів, тому він зберігається один раз і повторно використовується. Зовнішній стан надходить ззовні під час роботи програми та не зберігається у легковаговика. Шаблон зазвичай застосовується разом із фабрикою, яка забезпечує створення або повторне використання легковагових об’єктів, зберігаючи їх у пулі. Flyweight дозволяє суттєво зменшити кількість створюваних об’єктів і тим самим оптимізувати використання системних ресурсів. Його застосовують у графічних редакторах, текстових системах, ігрових рушіях, де потрібно представляти велику кількість однотипних елементів, наприклад символів тексту, дерев, частинок, графічних фігур чи інших повторюваних структур.

Шаблон «Interpreter»

Шаблон «Interpreter» належить до поведінкових шаблонів і призначений для опису граматики певної предметної мови та створення інтерпретатора, який може оцінювати вирази, побудовані

відповідно до цієї граматики. Основою шаблону є представлення кожного правила граматики окремим класом, який уміє інтерпретувати відповідну частину виразу. Зазвичай створюється абстрактний клас виразів і конкретні підкласи, що відповідають терміналам, нетерміналам або операціям. Інтерпретатор будує дерево виразів, у якому кожен вузол являє собою окремий фрагмент синтаксису. Обхід дерева дає змогу виконати логіку, визначену правилами мови. Interpreter підходить для простих мов, математичних виразів, обмежених граматик, конфігураційних правил і систем, де вирази потрібно часто аналізувати, але граматика не є надто складною. Його перевагою є гнучкість і легкість розширення граматики, а недоліком — погана масштабованість для складних мов.

Шаблон «Visitor»

Шаблон «Visitor» є поведінковим та використовується для відокремлення алгоритмів від структур даних, над якими вони виконуються. Його основна мета — дозволити додавати нові операції, не змінюючи класи елементів структури. Структура шаблону включає елементи, які мають метод прийняття відвідувача, та відвідувачів, які реалізують різні алгоритми обробки. Елемент передає себе об’єкту-відвідувачу, а той виконує відповідний метод, що відповідає типу елемента. Такий механізм забезпечує подвійну диспетчеризацію: вибір методу залежить від типу відвідувача та типу елемента. Visitor зручний у випадках, коли структура даних є стабільною, але алгоритми над нею змінюються часто. Наприклад, його застосовують для обходу дерев і графів, генерації звітів, аналізу синтаксичних дерев, оптимізації, перевірки, логування або виконання довільних додаткових операцій над складними структурами.

Хід виконання

Для розробленої системи архівації було обрано шаблон проєктування Visitor. Вибір цього патерну обумовлений специфікою структури даних у проєкті, яка складається з кількох різних типів елементів: архіву, файлів та частин архівів, всі ці об’єкти є елементами складної структури, які необхідно обробляти однаковим способом, не змінюючи їх внутрішню реалізацію. Нам було необхідно реалізувати перевірку контрольної суми, необхідно збирати всі файли та генерувати єдину суму та завдяки цьому шаблону можна легко додати інші обробники

Суть шаблону «visitor» полягає у тому, що він дозволяє операції над елементами без зміни структури конкретних елементів.

Для реалізації цього шаблону було виконано такі дії:

- Створено інтерфейс ArchiveElement, що описує метод accept() для передачі елементів відвідувачеві;
- Створено інтерфейс ArchiveVisitor, що описує метод visit(), що дозволяє відвідувачеві працювати з кожним об’єктом окремо;
- Створено клас ChecksumVisitor, що імплементується від ArchiveVisitor та реалізовує метод getChecksum(), що відповідає за генерування контрольної суми;

Переваги обраного шаблону:
- Зручний обхід складних структур;
- Легкість підтримання та доповнення коду;
- підтримка SOLID;

На рис. 1 наведена модифікована діаграма класів із представленням використання шаблону в реалізації системи

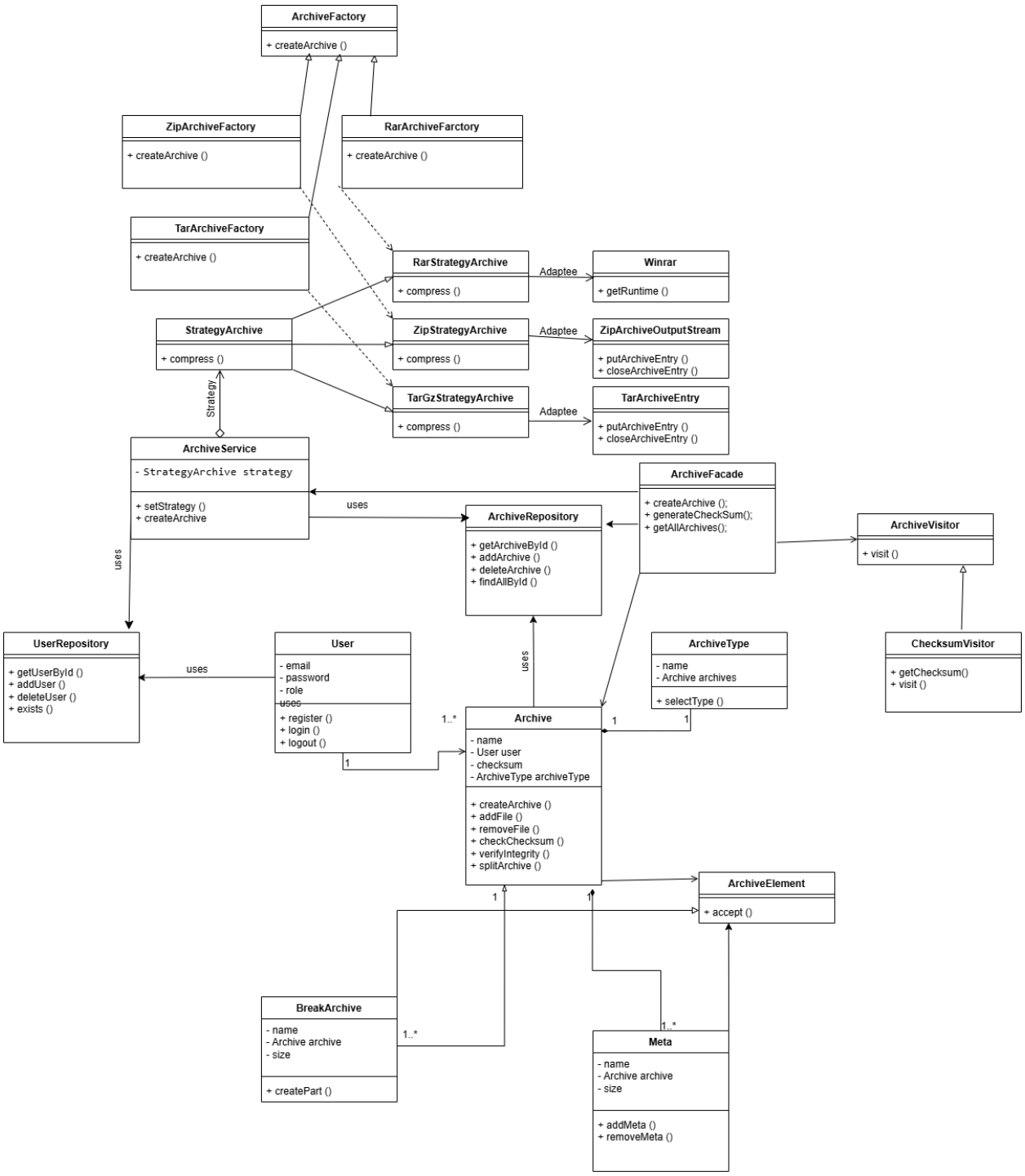


Рис. 1. Діаграма класів

Код програми

```
package com.example.demo.visitor;

import com.example.demo.model.Archive;

import com.example.demo.model.Meta;

import com.example.demo.model.BreakArchive;

public interface ArchiveVisitor {

    void visit(Archive archive);
```

```
    void visit(Meta meta);
    void visit(BreakArchive part);
}
package com.example.demo.visitor;
public interface ArchiveElement {
    void accept(ArchiveVisitor visitor);
}
package com.example.demo.visitor;
import com.example.demo.model.Archive;
import com.example.demo.model.Meta;
import com.example.demo.model.BreakArchive;
import java.security.MessageDigest;
public class ChecksumVisitor implements ArchiveVisitor {
    private final StringBuilder data = new StringBuilder();
    @Override
    public void visit(Archive archive) {
        data.append(archive.getName());
        data.append(archive.getArchiveTypeId());
        data.append(archive.getUserId());
    }
    @Override
    public void visit(Meta meta) {
        data.append(meta.getFileName());
        data.append(meta.getSize());
    }
    @Override
    public void visit(BreakArchive part) {
        data.append(part.getName());
        data.append(part.getSize());
    }
    public String getChecksum() {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(data.toString().getBytes());
```

```
            StringBuilder hex = new StringBuilder();
            for (byte b : hash) {
                hex.append(String.format("%02x", b));
            }
            return hex.toString();
        } catch (Exception e) {
            return "error";
        }
    }
}
```

Висновок

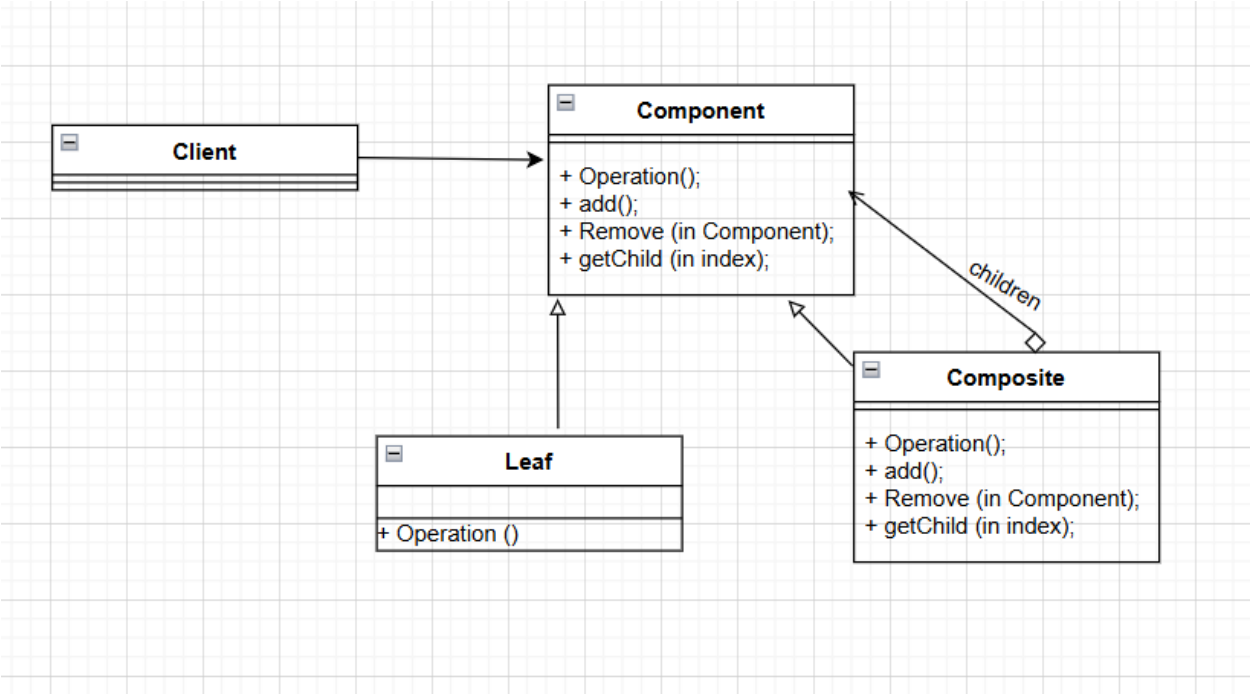
Під час виконання лабораторної роботи було реалізовано шаблон «Visitor» згідно теми «архіватор», під час реалізації було розроблено інтерфейс ArchiveElement, що описує метод accept() для передачі елементів відвідувачеві, інтерфейс ArchiveVisitor, що описує метод visit(), що дозволяє відвідувачеві працювати з кожним об’єктом окремо Створено клас ChecksumVisitor, що імплементується від ArchiveVisitor та реалізовує метод getChecksum(), що відповідає за генерування контрольної суми

Питання до лабораторної роботи

1. Яке призначення шаблону «Композит»?

Шаблон «Композит» призначений для роботи з деревоподібними структурами об’єктів, де окремі елементи та їхні групи мають поводитися однаково. Основна ідея полягає в тому, щоб клієнтський код не розрізняв прості об’єкти та складені об’єкти, оскільки вони реалізують спільний інтерфейс. Це забезпечує можливість рекурсивної побудови ієрархій та спрощує роботу з ними, бо клієнт обробляє будь-який елемент як цілісну сутність, незалежно від того, є він листком чи контейнером. Композит дозволяє створювати зручні, уніфіковані операції для виконання над структурою в цілому.

2. Нарисуйте структуру шаблону «Композит».



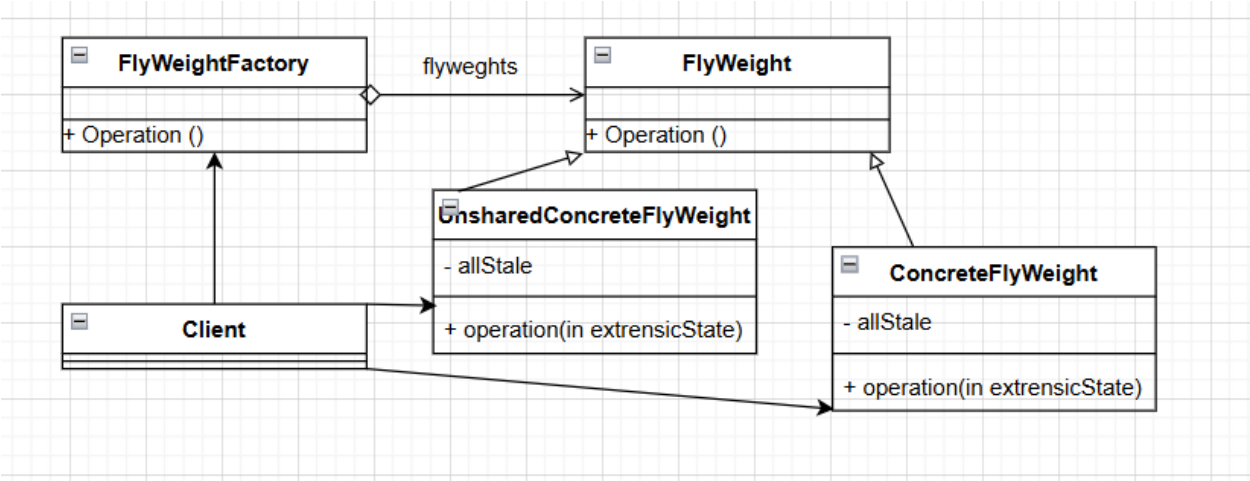
3. Які класи входять в шаблон «Композит», та яка між ними взаємодія?

У шаблон входить базовий компонент, який оголошує спільні методи для роботи як із простими, так і зі складеними об’єктами. Листовий клас реалізує поведінку одиничного елемента, що не має нащадків. Клас композита містить колекцію компонентів і реалізує додавання, видалення та виконання операцій шляхом виклику методів своїх дочірніх елементів. Взаємодія полягає в тому, що клієнт працює через інтерфейс компонента, не зважаючи на те, який конкретний тип об’єкта перед ним.

4. Яке призначення шаблону «Легковаговик»?

Шаблон «Легковаговик» призначений для мінімізації витрат пам’яті шляхом спільного використання об’єктів, які мають однакові внутрішні дані. Він розділяє стан об’єкта на внутрішній і зовнішній. Внутрішній стан є незмінним і може бути спільним для багатьох об’єктів, тоді як зовнішній стан передається ззовні під час виконання операцій. Це дозволяє значно скоротити кількість створюваних екземплярів і зменшити навантаження на пам’ять, особливо у великих структурах.

5. Нарисуйте структуру шаблону «Легковаговик».



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

До шаблону входять легковаговики з внутрішнім станом, фабрика легковаговиків, що створює або повторно використовує вже існуючі об’єкти, та клієнт, який працює з легковаговиками, передаючи їм зовнішній стан. Взаємодія полягає в тому, що клієнт звертається до фабрики, отримує посилання на вже існуючий легковаговий об’єкт, а під час виконання дій фабрика не бере участі — лише забезпечує спільне використання внутрішнього стану.

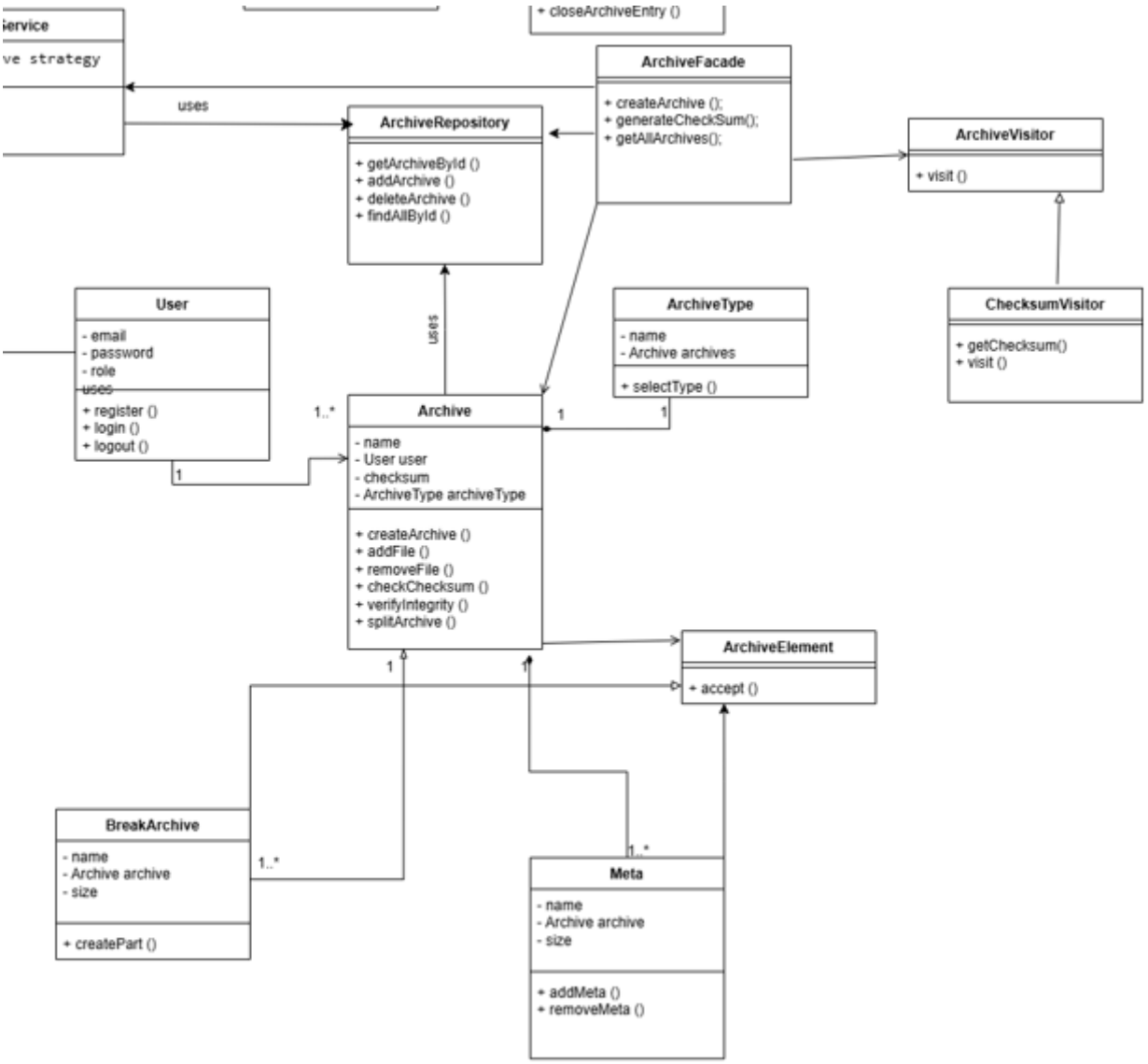
7. Яке призначення шаблону «Інтерпретатор»?

Шаблон «Інтерпретатор» призначений для опису граматики певної мови та побудови інтерпретатора, який може обробляти вирази цієї мови. Кожен елемент граматики представляється окремим класом, і за допомогою комбінування цих класів будується дерево виразів. Інтерпретатор дозволяє легко розширювати граматику і виконувати обчислення або аналіз побудованих виразів. Він використовується там, де часто потрібно виконувати однотипні операції над текстовими командами, математичними формулами, правилами або скриптами.

8. Яке призначення шаблону «Відвідувач»?

Шаблон «Відвідувач» призначений для відокремлення алгоритмів від структур даних, над якими вони виконуються. Це дозволяє додавати нові операції, не змінюючи класи елементів, над якими вони працюють. Кожен тип операції описується окремим відвідувачем, а елементи структури приймають цього відвідувача і передають себе йому для обробки. Такий підхід зручний у ситуаціях, коли структура фіксована, а операції над нею часто змінюються.

9. Нарисуйте структуру шаблону «Відвідувач».



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

До шаблону входять базовий інтерфейс відвідувача з методами для кожного класу елементів, конкретні реалізації відвідувачів, базовий інтерфейс елемента зі спеціальним методом прийняття відвідувача та конкретні елементи, що реалізують цей метод. Взаємодія полягає в тому, що елемент викликає метод відвідувача, передаючи йому себе, а відвідувач виконує свій алгоритм відповідно до типу елемента. Це дозволяє додавати нові обробки без зміни структури елементів.