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 №4
Технології розроблення програмного забезпечення
«Вступ до паттернів проектування»

Тема: Особиста бухгалтерія

Виконала:
студентка групи ІА-32
Коляда М. С.

Перевірила:
Мягкий М.Ю.

Київ 2025

Зміст

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» 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.

Теоретичні відомості

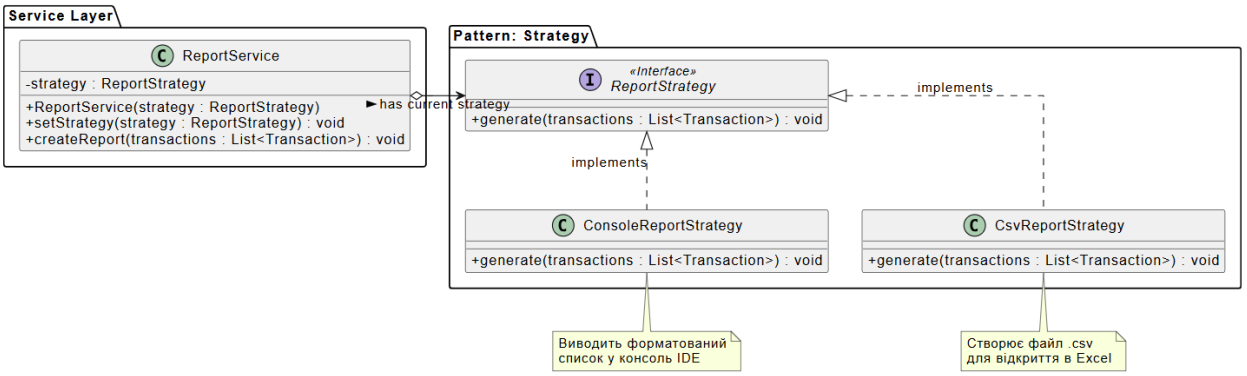
Шаблон «Strategy»

Призначення: Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує [6]. Даний шаблон дуже зручний у випадках, коли існують різні «політики» обробки даних. По суті, він дуже схожий на шаблон «State» (Стан), проте використовується в абсолютно інших цілях – незалежно від стану об'єкта відобразити різні можливі поведінки об'єкта (якими досягаються одні й ті самі або схожі цілі). Рішення: Коли ви використовуєте патерн «Стратегія», то схожі алгоритми виносяться з класа контекста в конкретні стратегії, за рахунок чого клас контексту стає чистіше і його легше супроводжувати.

Також одні і тіж самі стратегії можна використати з різними контекстами, що значно збільшує гнучкість вашої системи та зменшує кількість дублювань у коді. Контекст при цьому містить посилання на конкретну стратегію, а коли стратегію потрібно замінити, то замінюється об’єкт стратегії в полі Context.strategy. Важливою умовою є відносна простота інтерфейсу для алгоритмів стратегій. Якщо алгоритмам стратегій прийдеться передавати десятки параметрів, то це, скоріш за все, приведе до ускладнення системи та заплутаності коду. Якщо стратегії на вході будуть приймати об’єкт контексту, щоб отримувати з нього всі необхідні дані, то такі стратегії будуть прив’язані до конкретного контексту і їх не можна буде використати з іншим типом контексту. Приклад з життя: Ви їдете на роботу. Можна доїхати на автомобілі, на метро, або йти пішки. Тут алгоритм, як ви добираєтеся на роботу, є стратегією. В залежності від поточної ситуації ви вибираєте стратегію, що найбільше підходить в цій ситуації, наприклад, на дорогах великі пробки тоді ви їдете на метро, або метро тимчасово не ходить тоді ви їдете на таксі, або ви знаходитеся в 5 хвилинах ходьби від місця роботи і простіше добратися пішки. Переваги та недоліки: + Використовувані алгоритми можна змінювати під час виконання. + Реалізація алгоритмів відокремлюється від коду, що його використовує. + Зменшує кількість умовних операторів типу switch та if в контексті.

Хід роботи

Діаграма класів:



Опис реалізації патерну «Strategy»

Метою впровадження патерну була відмова від громіздких умовних конструкцій (if-else або switch) при виборі формату експорту даних, а також забезпечення можливості легкого додавання нових форматів (наприклад, PDF або JSON) у майбутньому без зміни основного коду програми.

Система підтримує дві стратегії генерації звітів:

1. Console Output (Консольний вивід): Швидкий перегляд транзакцій безпосередньо у вікні термінала (консолі IDE) для оперативної перевірки даних.
2. CSV Export (Експорт у файл): Формування файлу формату .csv, який дозволяє користувачеві відкрити фінансовий звіт у зовнішніх програмах, таких як Microsoft Excel або Google Sheets.

Механізм роботи: Клас ReportService виступає в ролі Контексту (Context). Він зберігає посилання на поточний об'єкт стратегії через інтерфейс ReportStrategy. Коли надходить запит на створення звіту (createReport), сервіс не виконує генерацію самостійно, а делегує це завдання поточному об'єкту стратегії. Залежно від того, яка стратегія зараз встановлена (встановлюється через сеттер), дані або виводятся на екран, або

записуються у файл. Зміна стратегії відбувається динамічно під час виконання програми.

Учасники патерну:

- ReportService (Context / Контекст) Клас бізнес-логіки, який виступає контекстом. Він зберігає посилання на екземпляр поточної стратегії в полі strategy. Сервіс не реалізує логіку форматування даних самостійно, а делегує її об'єкту стратегії через виклик методу generate(). Також містить метод setStrategy() для динамічної зміни способу генерації звіту під час виконання програми (наприклад, при натисканні відповідної кнопки в UI).
- ReportStrategy (Strategy / Інтерфейс Стратегії) Загальний інтерфейс, який визначає контракт поведінки для всіх можливих алгоритмів звітності. Він декларує метод generate(List<Transaction> transactions), реалізація якого буде специфічною для кожного конкретного формату.
- ConsoleReportStrategy (Concrete Strategy / Конкретна стратегія) Реалізує алгоритм виводу даних у консоль. У цій стратегії транзакції форматуються у вигляді текстової таблиці та виводяться через системний потік виводу (System.out). Використовується для швидкого перегляду без створення зайвих файлів.
- CsvReportStrategy (Concrete Strategy / Конкретна стратегія) Реалізує алгоритм експорту даних у файлову систему. Ця стратегія створює файл finance_report.csv, записує заголовки стовпців та ітерує список транзакцій, записуючи кожну з них у форматі, сумісному з Excel (розділеному крапкою з комою або комою). Реалізує роботу з потоками вводу-виводу (PrintWriter, FileWriter).

Фрагменти програмного коду:

ReportStrategy.java

```
package ia32.koliada.finance.report;

import ia32.koliada.finance.entity.Transaction;
import java.util.List;

public interface ReportStrategy {
    void generate(List<Transaction> transactions);
}
```

CsvReportStrategy.java

```
package ia32.koliada.finance.report;

import ia32.koliada.finance.entity.Transaction;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

public class CsvReportStrategy implements ReportStrategy {
    @Override
    public void generate(List<Transaction> transactions) {
        String filename = "finance_report.csv";
        try (PrintWriter writer = new PrintWriter(new FileWriter(filename))) {
            writer.println("Date;Description;Amount");
            for (Transaction t : transactions) {
                writer.printf("%s;%s;%s%n",
                    t.getFormattedDate(),
                    t.getDescription(),
                    t.getAmount());
            }
            System.out.println(">>> Звіт успішно збережено у файл: " +
filename);
        } catch (IOException e) {
            System.err.println("Помилка запису файлу: " + e.getMessage());
```

```
        }
    }
}
```

ConsoleReportStrategy.java

```
public class ConsoleReportStrategy implements ReportStrategy {
    @Override
    public void generate(List<Transaction> transactions) {
        System.out.println("----- ФІНАНСОВИЙ ЗВІТ -----");
        for (Transaction t : transactions) {
            System.out.printf("%s | %s | %s грн%n",
                    t.getFormattedDate(), t.getDescription(), t.getAmount());
        }
        System.out.println("--------------------------");
    }
}
```

Вихідний код:

https://github.com/mariiakoliada/TRPZ/tree/master/src/ia32/koliada/finance/report

Питання до лабораторної роботи

1. Що таке шаблон проєктування?

Шаблон проєктування – це типове, перевірене часом вирішення певної проблеми, яка часто виникає при проєктуванні архітектури програмного забезпечення. Це не готовий шматок коду, який можна просто скопіювати, а опис схеми взаємодії класів та об'єктів для вирішення задачі

2. Навіщо використовувати шаблони проєктування?

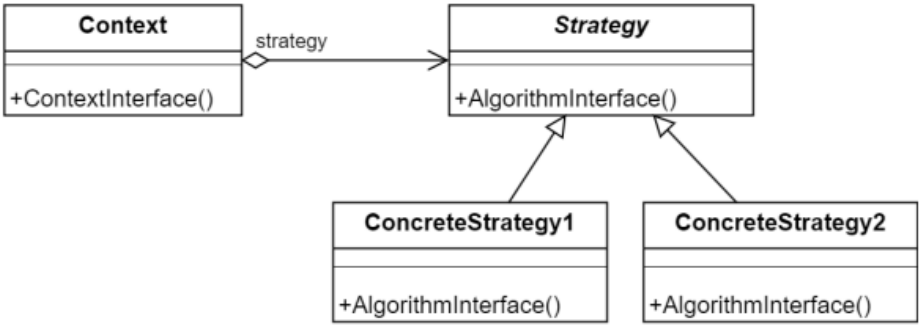
Застосування патернів проєктування забезпечує можливість багаторазового використання перевірених архітектурних рішень. Це сприяє підвищенню прозорості та зрозумілості моделі системи, що дозволяє детальніше опрацювати її структуру. Крім того, шаблони збільшують адаптивність

програмного забезпечення до змін у вимогах, полегшують процес супроводу коду та формують уніфіковану термінологію для ефективної комунікації в команді розробників.

3. Яке призначення шаблону «Стратегія»?

Він визначає сімейство алгоритмів, інкапсулює кожен з них у окремий клас і робить їх взаємозамінними. Стратегія дозволяє змінювати алгоритми незалежно від клієнтів, які їх використовують (наприклад, змінити спосіб сортування чи формат експорту файлу "на льоту").

4. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

Context (Контекст): Клас, якому потрібно виконати певну роботу. Він зберігає посилання на інтерфейс Strategy і спілкується з ним, не знаючи конкретної реалізації.

Strategy (Стратегія): Спільний інтерфейс для всіх алгоритмів.

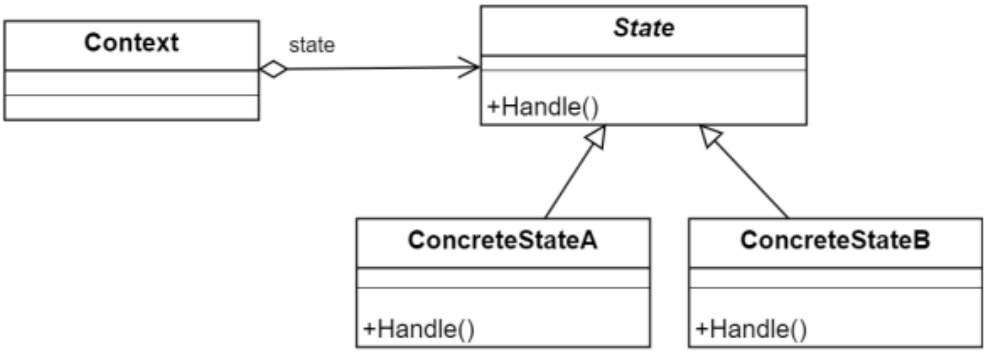
ConcreteStrategy (Конкретна Стратегія): Класи, що реалізують конкретний алгоритм.

Взаємодія: Клієнт створює об'єкт конкретної стратегії та передає його в Контекст. Контекст делегує виконання роботи об'єкту стратегії.

6. Яке призначення шаблону «Стан»?

Дозволяє об'єкту змінювати свою поведінку при зміні його внутрішнього стану.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Context (Контекст): Об'єкт, поведінка якого змінюється. Зберігає посилання на поточний Стан.

State (Стан): Інтерфейс для інкапсуляції поведінки.

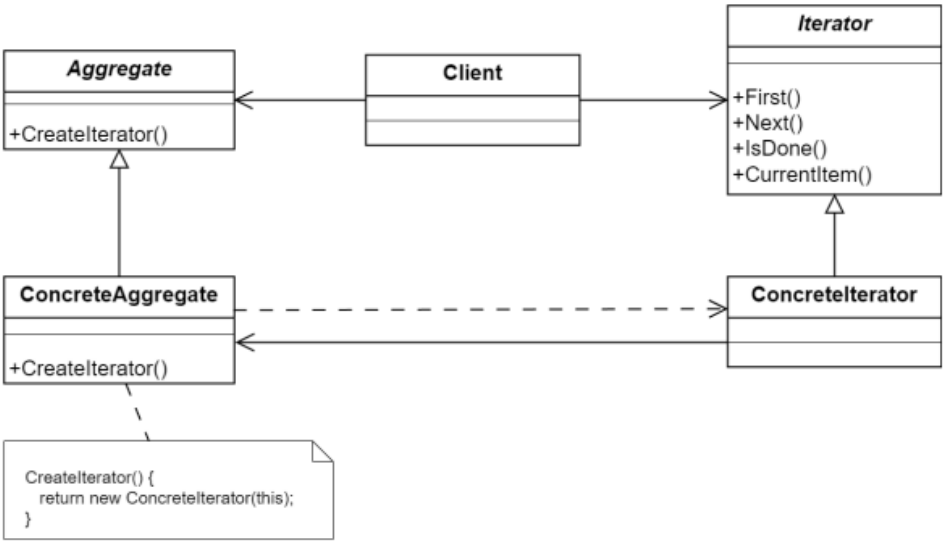
ConcreteState (Конкретний стан): Реалізує поведінку, специфічну для певної стадії життя об'єкта.

Взаємодія: Контекст делегує виконання операцій об'єкту поточного стану. Самі стани можуть ініціювати зміну стану в Контексті (перехід від A до B).

9. Яке призначення шаблону «Ітератор»?

Надає спосіб послідовного доступу до всіх елементів складеного об'єкта (колекції), не розкриваючи його внутрішнього представлення (масив це, список чи дерево).

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

– Aggregate (Колекція): відповідає за зберігання даних.

– Iterator (Ітератор): відповідає за прохід по колекції, відстежує стан обходу та поточну позицію.

Клієнт отримує ітератор від колекції і використовує його для перебору елементів у циклі.

12. В чому полягає ідея шаблона «Одинак»?

Гарантувати, що клас має лише один екземпляр, і надати до нього глобальну точку доступу.

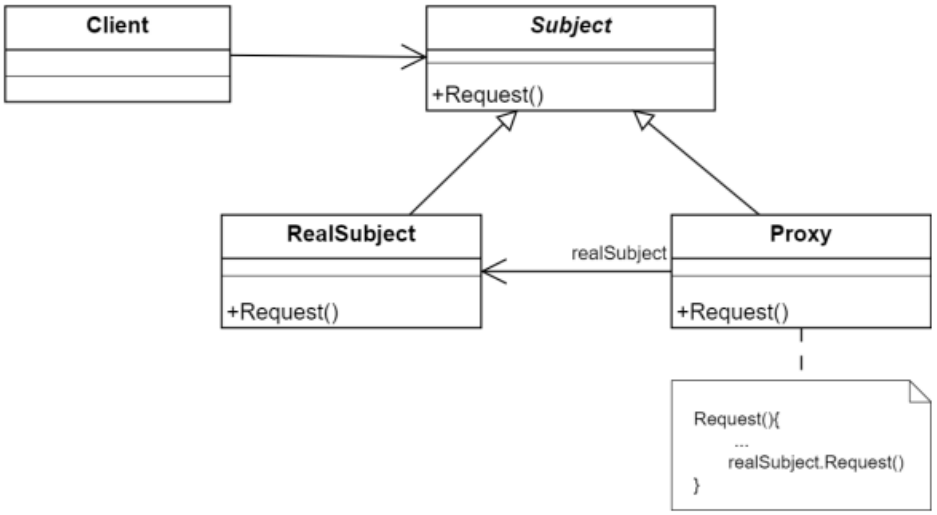
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Шаблон "Одинак" часто класифікують як анти-патерн через низку архітектурних недоліків. По-перше, він фактично створює глобальний стан програми, зміни якого важко контролювати та відстежувати, що знижує надійність системи. По-друге, наявність жорстких залежностей від глобального об'єкта значно ускладнює процес модульного тестування (Unit testing). Крім того, цей шаблон порушує принцип єдиної відповідальності (Single Responsibility Principle), оскільки клас змушений відповідати і за свою бізнес-логіку, і за керування власним життєвим циклом.

14. Яке призначення шаблону «Проксі»?

Надає об'єкт-замісник, який контролює доступ до іншого об'єкта. Це дозволяє перехоплювати виклики до оригінального об'єкта (наприклад, для лінивої ініціалізації, перевірки прав доступу, кешування або логування).

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Subject: Спільний інтерфейс для реального об'єкта та замісника.

RealSubject: Об'єкт, який виконує реальну роботу.

Proxy: Зберігає посилання на RealSubject. Реалізує той самий інтерфейс Subject.

Клієнт працює з Proxy як з реальним об'єктом. Proxy може виконати щось "до" або "після" виклику (наприклад, перевірити права) і, якщо потрібно, передає виклик RealSubject.

Висновок: під час виконання лабораторної роботи, було вивчено структуру та принципи роботи патернів проєктування «Singleton», «Iterator», «Proxy», «State» та «Strategy». Набуто практичних навичок їх застосування при розробці програмної системи. Зокрема, успішна реалізація шаблону «Стратегія» дозволила створити гнучкий механізм генерації звітів (у консоль та файл), що зробило архітектуру системи модульною та зручною для подальшого розширення.