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

Тема: Вступ до патернів проектування.

Виконала
студентка групи ІА–32:
Слюсарева А.А.

Київ 2025

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 4

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

4.1. 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.

4.2. Теоретичні відомості

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. 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).

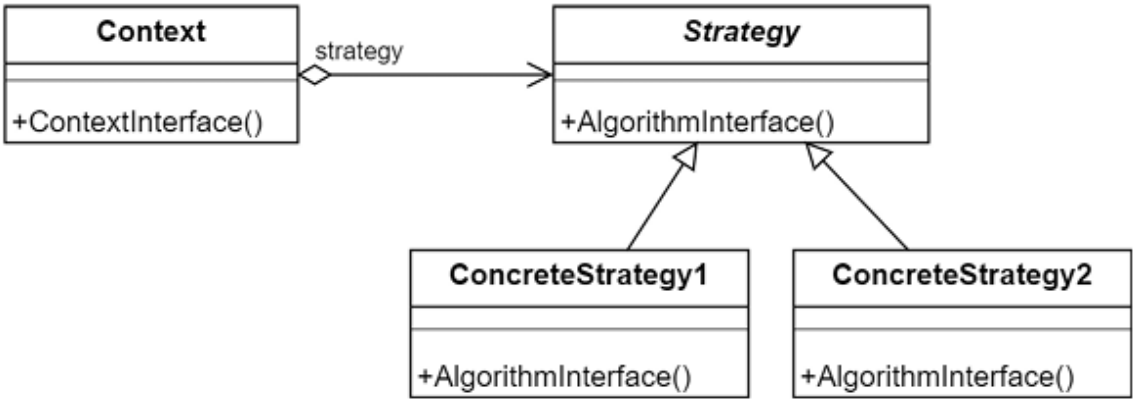


Рисунок 4.5. Структура патерну Стратегія.

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.

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.

- Надмірна складність, якщо у вас лише кілька невеликих алгоритмів.

- Під час виклику алгоритму, клієнтський код має враховувати різницю між стратегіями.

4.3 Хід роботи

Діаграма класів:

Рис 1. Діаграма класів системи

Фрагменти коду:

```
import { Injectable } from '@angular/core';
import { IFormatStrategy } from
'../models/format-strategy.model';
import { MinifyStrategy, PrettifyStrategy } from
'./format-strategies';

@Injectable({
 providedIn: 'root'
})
export class FormatService {
 private strategy: IFormatStrategy;

 constructor() {
   this.strategy = new PrettifyStrategy();
 }

 setStrategy(strategy: IFormatStrategy): void {
   this.strategy = strategy;
 }

 format(json: string): string {
   return this.strategy.format(json);
 }

 minify(json: string): string {
   this.setStrategy(new MinifyStrategy());
   return this.format(json);
```

```
  }

  prettify(json: string): string {
    this.setStrategy(new PrettifyStrategy());
    return this.format(json);
  }
}
```

```
import { IFormatStrategy } from
'../models/format-strategy.model';

export class MinifyStrategy implements
IFormatStrategy {
  format(json: string): string {
    try {
      const parsed = JSON.parse(json);
      return JSON.stringify(parsed);
    } catch (error) {
      throw new Error('Invalid JSON format');
    }
  }
}

export class PrettifyStrategy implements
IFormatStrategy {
  format(json: string): string {
    try {
      const parsed = JSON.parse(json);
      return JSON.stringify(parsed, null, 2);
```

```
    } catch (error) {
      throw new Error('Invalid JSON format');
    }
  }
}
```

У класі FormatService використовується патерн "Стратегія", який дозволяє задавати різні способи форматування JSON без зміни логіки самого сервісу. Об’єкти MinifyStrategy та PrettifyStrategy реалізують спільний інтерфейс IFormatStrategy, тому сервіс може динамічно перемикатися між цими алгоритмами.
Методи minify() та prettify() просто змінюють поточну стратегію, а вся робота виконується обраним об’єктом-стратегією.

4.4 Висновок

Під час даної роботи було реалізовано патерн Стратегія у додатку JSON Tool.

4.5 Контрольні запитання

1. **Що таке шаблон проєктування?**
   Шаблон проєктування – це типове, багаторазове рішення певної задачі проєктування у програмуванні. Він описує загальну схему, яку можна адаптувати під конкретну проблему.

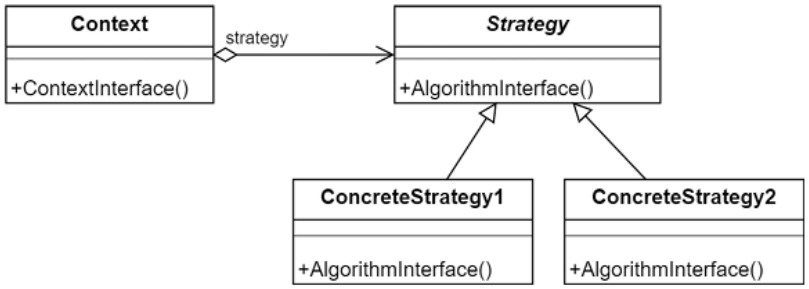
2. **Навіщо використовувати шаблони проєктування?**
   Шаблони проєктування допомагають спростити створення архітектури програми, підвищують гнучкість і зрозумілість коду, дозволяють уникнути

типових помилок та повторного винаходу рішень.

3. **Яке призначення шаблону «Стратегія»?**

   Шаблон «Стратегія» дозволяє динамічно змінювати алгоритми під час виконання програми, інкапсулюючи їх у окремі класи зі спільним інтерфейсом.

4. **Нарисуйте структуру шаблону «Стратегія».**



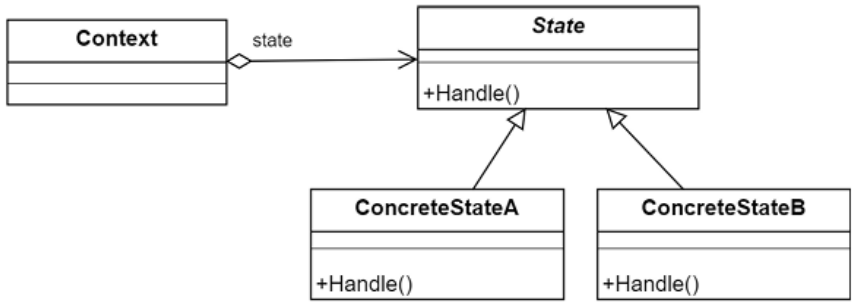
5. **Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?**

   До шаблону входять: Контекст, Інтерфейс Стратегії та Конкретні Стратегії. Контекст містить посилання на об’єкт стратегії та викликає його методи, а стратегії реалізують різні алгоритми.

6. **Яке призначення шаблону «Стан»?**

   Шаблон «Стан» дозволяє об’єкту змінювати свою поведінку залежно від внутрішнього стану, при цьому самі стани оформлені як окремі класи.

7. **Нарисуйте структуру шаблону «Стан».**



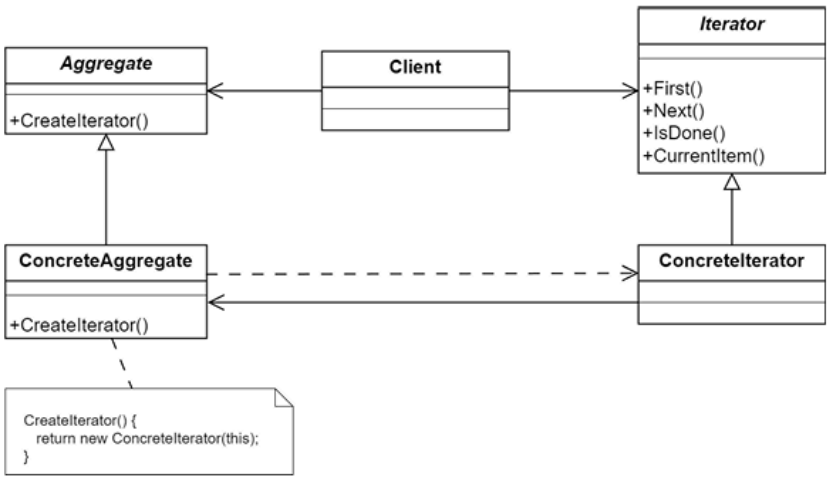
8. **Які класи входять в шаблон «Стан», та яка між ними взаємодія?**

   До шаблону входять: Контекст, Інтерфейс Стану, Конкретні Стани. Контекст делегує поведінку активному стану, а стани можуть змінювати стан контексту.

9. **Яке призначення шаблону «Ітератор»?**

   «Ітератор» забезпечує послідовний доступ до елементів колекції без розкриття її внутрішньої структури.

10. **Нарисуйте структуру шаблону «Ітератор».**



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

    Основні класи: Колекція, Інтерфейс Ітератора, Конкретний Ітератор.

Колекція створює і повертає ітератор, а ітератор надає методи для переміщення по елементах.

12. **В чому полягає ідея шаблона «Одинак»?**

Ідея шаблону «Одинак» полягає в тому, що створюється лише один екземпляр класу, доступний глобально через статичний метод.

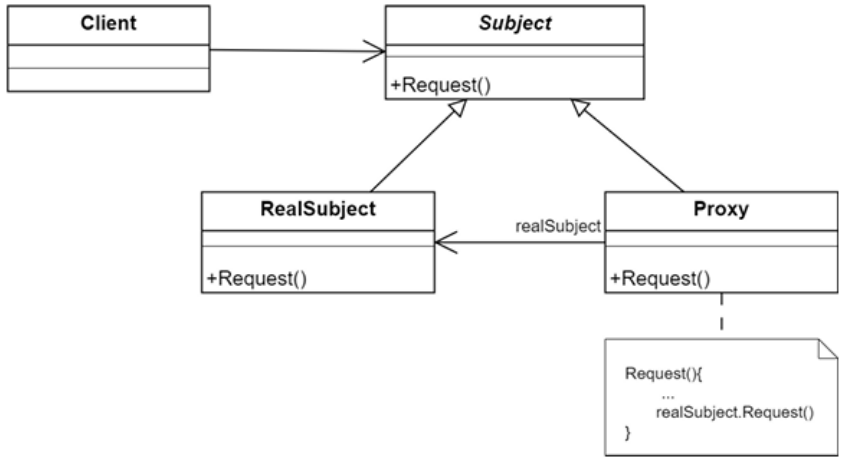
13. **Чому шаблон «Одинак» вважають «анти-шаблоном»?**

Його критикують через глобальний стан, складність тестування та небезпеку непомітних побічних ефектів у великих системах.

14. **Яке призначення шаблону «Проксі»?**

Шаблон «Проксі» створює сурогатний об’єкт, який контролює доступ до реального об’єкта, додаючи кешування, ліниву ініціалізацію, безпеку тощо.

15. **Нарисуйте структуру шаблону «Проксі».**



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

До шаблону входять: Інтерфейс Компонента, Реальний Об’єкт, Проксі. Клієнт працює з проксі як із справжнім об’єктом, а проксі контролює доступ до реального об’єкта.