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

ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Виконав
студент групи ІА–22:
Щаблевський Д. Е.
Перевірив:
Мягкий Михайло Юрійович

Київ 2024

Зміст

Тема: шаблони «singleton», «iterator», «proxy», «state», «**strategy**».

Мета: ознайомитися з шаблонами проектування «singleton», «iterator», «proxy», «state», «**strategy**», та набути практичних навичок їх застосування. Реалізувати частину функціоналу програми за допомогою одного з розглянутих шаблонів для досягнення конкретних функціональних можливостей та забезпечення ефективної взаємодії між класами.

Хід роботи

..28 JSON Tool (ENG) (strategy, command, observer, template method, flyweight)

Display JSON schema with syntax highlight. Validate JSON schema and display errors. Create user friendly table\list box\other for read and update JSON schema properties metadata (description, example, data type, format, etc.). Auto save\restore when edit, maybe history. Can check JSON value by schema (Put schema and JSON = valid\invalid, display errors). Export schema as markdown table. JSON to "flat" view.

Короткі теоретичні відомості

Шаблони проєктування представляють собою стандартизовані підходи до вирішення типових задач, які часто зустрічаються при створенні інформаційних систем. Вони включають в себе опис проблеми, ефективне рішення та рекомендації щодо застосування в різних контекстах. Завдяки загальновизнаним назвам, шаблони легко розпізнаються та багаторазово використовуються.

Ключовою особливістю роботи з шаблонами є ретельне моделювання предметної області, що дозволяє точно сформулювати проблему і підібрати найкращий шаблон. Використання таких рішень дає розробнику ряд переваг: вони допомагають зробити систему більш організованою, зрозумілою для вивчення та легкою у розширенні. Крім того, це підвищує гнучкість системи до змін і сприяє її простішій інтеграції з іншими рішеннями.

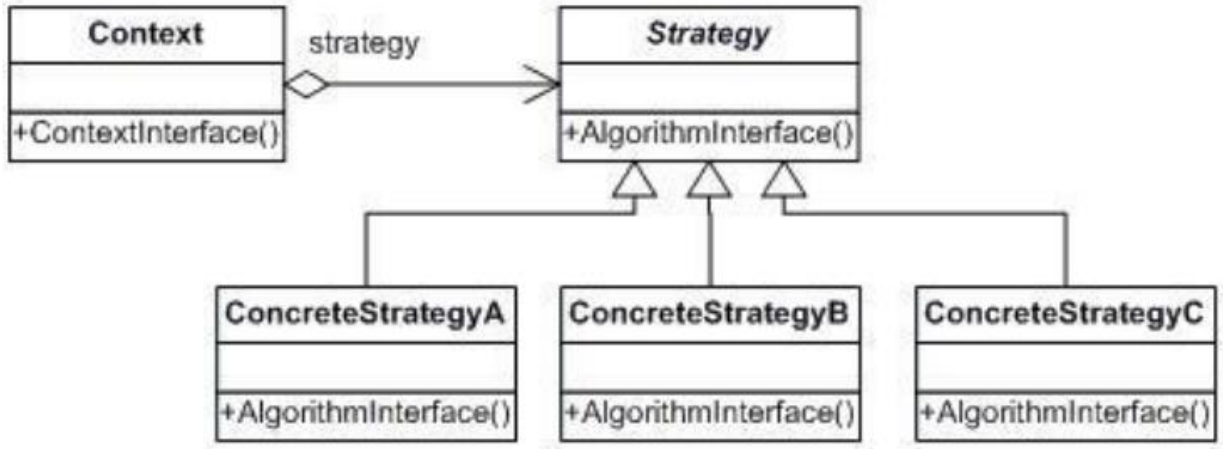
Таким чином, шаблони проєктування є перевіреними часом інструментами, які широко застосовуються в різних організаціях і проєктах для побудови ефективної архітектури в заданих умовах.

Навіщо використовувати шаблони ?

Шаблони проєктування сприяють зменшенню часу і зусиль на створення архітектури, надаючи системі важливі властивості, як-от гнучкість і

адаптованість, а також спрощують її підтримку. Їх застосування полегшує комунікацію в команді, оскільки спільне розуміння шаблонів робить архітектуру зрозумілою для всіх учасників проєкту. Шаблони дозволяють уникнути "винаходу велосипеда", використовуючи перевірені практики, визнані спільнотою розробників, що економічно вигідно для бізнесу. Водночас їх використання повинно бути обґрунтованим, адже вони не є універсальним рішенням для всіх ситуацій.

Шаблон “Strategy”



Призначення патерну Strategy

Шаблон «Strategy» (Стратегія) дозволяє змінювати алгоритм поведінки об'єкта на інший, що досягає ту ж мету іншим способом. Наприклад, різні алгоритми сортування, кожен з яких реалізований у окремому класі та може взаємозамінюватися в об'єкті. Цей шаблон зручний для реалізації різних «політик» обробки даних.

Проблема

Стратегія потрібна, якщо:

- Необхідно забезпечити можливість вибору з декількох алгоритмів для виконання певної задачі.

- Потрібно легко змінювати чи додавати нові алгоритми без зміни основного коду.

Рішення

Стратегія використовується для реалізації змінних способів обробки даних, дозволяючи вибір відповідного алгоритму під час виконання програми. Вона підходить для управління різними методами сортування, обробки запитів або роботи з конфігураціями, забезпечуючи легку заміну або розширення можливостей без модифікації існуючого коду.

Реалізувати не менше 3-х класів відповідно до обраної теми

Структура проекту з реалізованими класами зображена на рисунку 1.

Рисунок 1 – структура класів

Опис класів

В ході виконання лабораторної роботи були реалізовані наступні класи:

- **FormatterStrategy**
  - Інтерфейс, що визначає метод format(text: string): string.
  - Використовується для створення взаємозамінних стратегій форматування тексту.
- **FormatterStrategyService**

- Клас для роботи зі стратегіями форматування.
- Містить логіку для використання конкретної стратегії форматування.

- **CompactFormatterStrategy**
  - Реалізація інтерфейсу FormatterStrategy.
  - Виконує компактне форматування JSON-строк, прибираючи зайві пробіли та форматування.
- **SmartFormatterStrategy**
  - Реалізація інтерфейсу FormatterStrategy.
  - Виконує "оптимальне" форматування JSON-строк із додаванням відступів для покращення читабельності.

Реалізувати один з розглянутих шаблонів за обраною темою.

```
import { FormatterStrategy } from "../services/formatter-strategy.service";

export class CompactFormatterStrategy implements FormatterStrategy {
  format(jsonString: string): string {
    try {
      const jsonObject = JSON.parse(jsonString);
      return JSON.stringify(jsonObject);
    } catch (error) {
      throw new Error('Invalid JSON string');
    }
  }
}
```

```
import { FormatterStrategy } from "../services/formatter-strategy.service";

export class SmartFormatterStrategy implements FormatterStrategy {
  format(jsonString: string): string {
    try {
      const jsonObject = JSON.parse(jsonString);
      return JSON.stringify(jsonObject, null, 2); // Pretty print with 2 spaces
    } catch (error) {
      throw new Error('Invalid JSON string');
    }
  }
}
```

```
import { LoggerService } from "./logger.service";

export interface FormatterStrategy {
  format(text: string): string;
}

export class FormatterStrategyService implements FormatterStrategy {
  private readonly formatterStrategy: FormatterStrategy;
  private loggerService: LoggerService;

  constructor(
    formatterStrategy: FormatterStrategy,
    loggerService: LoggerService,
  ) {
    this.formatterStrategy = formatterStrategy;
    this.loggerService = loggerService;
  }

  format(text: string) {
    this.loggerService.log('Strategy', 'Using Formatter Strategy');

    return this.formatterStrategy.format(text);
  }
}
```

Рисунок 2 – реалізований шаблон «Стратегія»

У класі **FormatterStrategyService** використання конкретної стратегії визначається під час створення екземпляра. Завдяки цьому, замість створення жорстко закріпленої логіки, сервіс працює з будь-якою реалізацією інтерфейсу FormatterStrategy. Це дозволяє динамічно змінювати поведінку програми без зміни основного коду. Такий підхід реалізує принцип шаблону «Стратегія» і забезпечує гнучкість у роботі з різними алгоритмами форматування.

Проблема, яку допоміг вирішити шаблон «Стратегія»

Застосування патерну «**Стратегія**» вирішує проблему жорсткої прив’язки до одного алгоритму форматування. Раніше, якщо була потреба в іншому способі обробки даних, доводилося змінювати існуючий код, що могло призвести до помилок. Тепер всі алгоритми форматування ізольовані в окремих класах і легко взаємозамінні.

Переваги застосування патерну «Стратегія»

1. **Гнучкість і розширюваність**
   Стратегії форматування (**CompactFormatterStrategy** і **SmartFormatterStrategy**) реалізують спільний інтерфейс, що дозволяє легко додавати нові алгоритми форматування без зміни існуючого коду.
2. **Розділення відповідальності**
   Алгоритми форматування ізольовані в окремих класах, що робить код чистішим і легшим для тестування. Основний сервіс **FormatterStrategyService** відповідає тільки за виклик потрібної стратегії.
3. **Єдиний підхід до форматування**
   Використання одного сервісу з різними стратегіями дозволяє централізовано керувати логікою обробки тексту.
4. **Уникнення дублювання коду**
   Замість реалізації різних алгоритмів форматування в одному класі, кожна стратегія відповідає за свою унікальну реалізацію, що виключає дублювання.

Діаграма класів для паттерну «Стратегія»

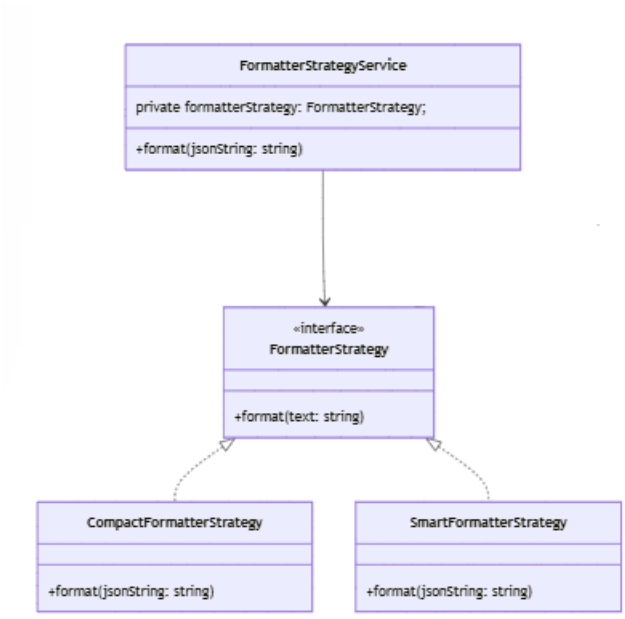


Рисунок 3 – діаграма класів

Посилання на репозиторій з кодом: https://github.com/neodavis/json-tool

Висновок: У ході виконання лабораторної роботи було реалізовано патерн проєктування Strategy для функціоналу форматування вводу користувача. Шаблон «Стратегія» дозволив забезпечити динамічний вибір алгоритмів форматування тексту під час виконання програми. Це зробило систему модульною, гнучкою та легкою для підтримки.