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

Тема: «Вступ до паттернів проектування.»

Виконала:

Студентка групи ІА-34

Сизоненко А.О

Перевірив:

Мягкий Михайло Юрійович

Київ-2025

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

Тема роботи: Текстовий редактор (strategy, command, observer, template method, flyweight, SOA)

Текстовий редактор повинен вміти розпізнавати текстові файли в будь-якій кодуванні, мати розширені функції редагування: макроси, сніппети, підказки, закладки, перехід на рядок / сторінку, підсвічування синтаксису (для однієї мови програмування або розмітки на розсуд студента).

Теоретичні відомості

Шаблон (патерн) проєктування – це формалізований опис часто зустрічаючогося рішення в розробці програмних систем із рекомендаціями щодо його застосування. Використання патернів спрощує моделювання системи, підвищує гнучкість, зрозумілість архітектури та стійкість до зміни вимог. Патерни дозволяють повторно використовувати перевірені рішення та слугують уніфікованим словником для спілкування розробників.

Патерн **Singleton** забезпечує наявність лише одного екземпляра класу та глобальну точку доступу до нього, що зручно для об’єктів конфігурацій або сеансів зв’язку. Він гарантує унікальність об’єкта, але порушує принцип єдиної відповідальності та ускладнює тестування.

Патерн **Iterator** дозволяє послідовно обходити елементи колекції без розкриття її внутрішньої структури. Це спрощує класи зберігання даних і дає можливість реалізувати різні способи обходу, але може бути зайвим для простих списків.

Патерн **Proxy** створює проміжний об’єкт-замінник, що контролює доступ до реального об’єкта та додає додаткову логіку. Це дозволяє оптимізувати роботу та керувати життєвим циклом об’єкта без зміни клієнтського коду, проте може знижувати продуктивність.

Патерн **State** дозволяє змінювати поведінку об’єкта при зміні його внутрішнього стану. Кожен стан реалізується окремим класом, а контекст делегує виклики стану. Це ізолює специфічну логіку та спрощує додавання нових станів, але ускладнює сам контекст.

Патерн **Strategy** дає можливість динамічно змінювати алгоритм поведінки об’єкта. Алгоритми винесені в окремі класи, що робить контекст чистішим і гнучким, дозволяє повторно використовувати стратегії та зменшує кількість умовних операторів у контексті. Основні недоліки – надмірна складність при невеликій кількості алгоритмів та необхідність враховувати вибір стратегії в клієнтському коді.

Хід роботи

1. Діаграма класів для реалізації шаблону стратегія.

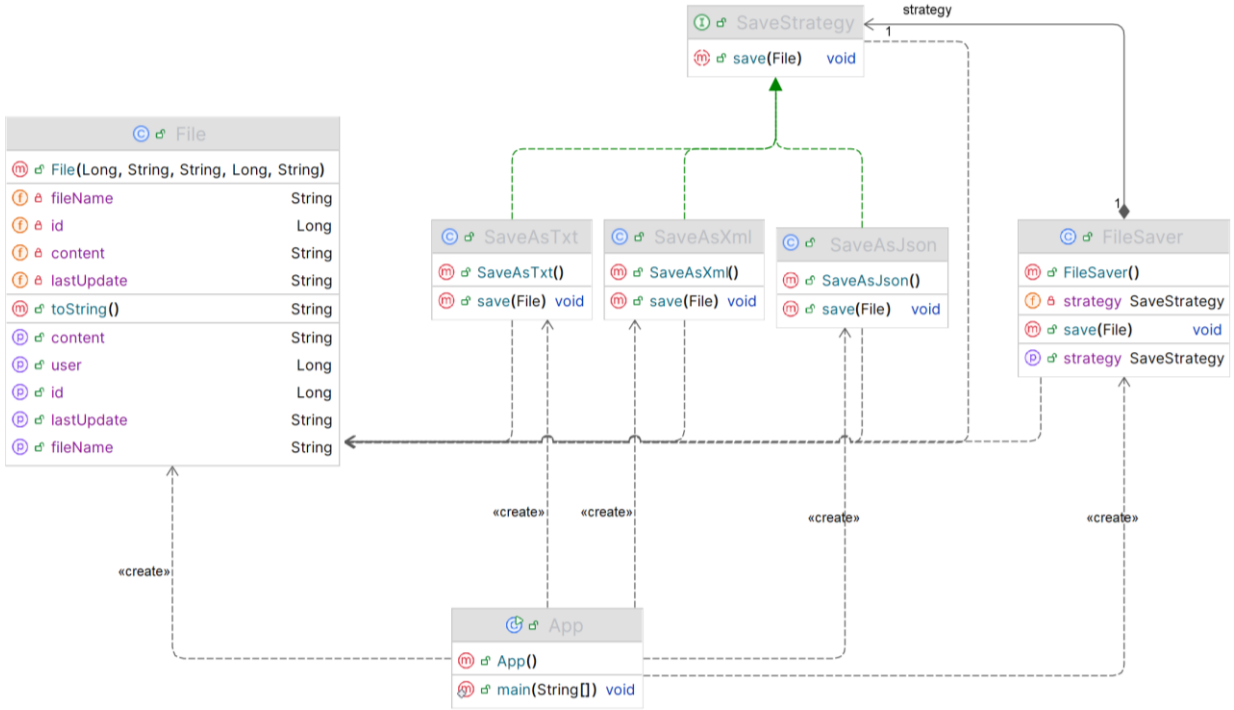

Рисунок 1 – Діаграма класів

Діаграма класів

1. File - модель, яка представляє файл у системі. Містить властивості файлу: назву, вміст, ідентифікатор власника та дату останнього редагування. Використовується для передачі даних між компонентами системи.
2. FileSaver - клас-контекст, який використовує стратегії збереження. Дозволяє встановлювати потрібну стратегію під час виконання. Забезпечує гнучкість і розширюваність системи.
3. SaveStrategy - інтерфейс стратегії збереження файлу. Визначає єдине правило (метод void save(File file)), яке мають реалізувати конкретні способи збереження. Забезпечує можливість гнучкої зміни формату збереження без зміни основного коду.
4. SaveAsTxt, SaveAsXml та SaveAsJson – конкретні стратегії для збереження файлу у текстовому форматі, в форматі json та xml. Ці стратегії реалізують метод save і забезпечують конкретний алгоритм збереження файлу.
5. App - головний клас програми. Створює файли та об’єкт FileSaver, встановлює конкретну стратегію збереження та виконує збереження файлу.

Приклад реалізації шаблону:

1. Ініціалізація об’єкта файлу
    - Створюється об’єкт класу File з відповідними властивостями: fileName, content, userid та lastUpdate.
2. Створення контексту та вибір стратегії

- Створюється об’єкт FileSaver, який виступає контекстом.
- Через метод setStrategy() вибирається конкретна стратегія збереження файлу:
  - Спочатку застосовується SaveAsTxt, щоб зберегти файл у форматі .txt.
  - Потім використовується SaveAsJSON, щоб зберегти файл у форматі .json.
  - Далі застосовується SaveAsXml, щоб зберегти файл у форматі .xml.

3. Виконання стратегії
   - Викликається метод saveFile() у FileSaver, який передає об’єкт File у відповідну стратегію:
     - SaveAsTxt — зберігає вміст файлу у простому текстовому форматі.
     - SaveAsJson — конвертує вміст файлу у JSON та зберігає його.
     - SaveAsXml — форматує вміст у XML та зберігає файл.

```
public class App {
    public static void main(String[] args) {
        File file = new File( id: 1L,
                fileName: "example",
                content: "Це тестовий контент файлу",
                userid: 100L,
                lastUpdate: "2025-11-03");

        FileSaver saver = new FileSaver();

        saver.setStrategy(new SaveAsTxt());
        saver.save(file);

        saver.setStrategy(new SaveAsJson());
        saver.save(file);

        saver.setStrategy(new SaveAsXml());
        saver.save(file);
    }
}
```

Рисунок 2 – код класу App

Результат виконання програми:

```
Збережено як TXT: example.txt
Файл збережено як JSON: example.json
Збережено як XML: example.xml
```

Рисунок 3 – вивід повідомлення в консолі

example.json

```json
{
    "filecontent": "Це тестовий контент файлу",
    "lastUpdate": "2025-11-03",
    "name": "example",
    "id": 1,
    "userid": 100
}
```

Рисунок 4 – файл збережений у форматі json

example.txt

```
Це тестовий контент файлу
```

Рисунок 5 – файл збережений у форматі txt

example.xml

```xml
<file>
  <id>1</id>
  <name>example</name>
  <userId>100</userId>
  <lastUpdate>2025-11-03</lastUpdate>
  <content>Це тестовий контент файлу</content>
</file>
```

Рисунок 6 – файл збережений у форматі xml

Переваги використання шаблону Стратегія для моєї теми проекту:

1. Гнучкість і розширюваність
    - Легко додати новий формат збереження (наприклад PDF або Markdown), не змінюючи існуючий код FileSaver чи інших стратегій.
    - Можна динамічно змінювати стратегію під час виконання програми.
2. Відокремлення різних способів збереження
    - Логіка збереження для TXT, JSON, XML ізольована в окремих класах.
    - Це робить код чистим і легким для підтримки.
3. Підтримка принципу відкритості/закритості (OCP)

- Система відкрита для розширення (додавання нових стратегій) і закрита для модифікації існуючого коду.

4. Уніфікований інтерфейс збереження
    - FileSaver завжди використовує один метод saveFile(), незалежно від формату, що спрощує використання.

Недоліки використання патерну Стратегія

1. Надмірна складність.
   Якщо в системі використовується лише кілька форматів збереження (наприклад TXT і JSON).
2. Клієнтський код має враховувати вибір стратегії, тобто потрібно явно встановлювати потрібну стратегію перед збереженням файлу.
    - (хоча може бути перевагою, оскільки дає повний контроль над тим, як саме буде збережений файл)

Висновок: у роботі реалізовано патерн Стратегія для збереження файлів у різні формати (TXT, JSON, XML). Це дозволяє відокремити логіку збереження від моделі файлу та контексту, спрощує додавання нових форматів і тестування стратегій. Основним недоліком є збільшення кількості класів та необхідність вибору стратегії перед збереженням, але це дає повний контроль над форматом файлу. Було створено діаграму класів, яка наочно демонструє роботу шаблону та взаємодію основних компонентів системи. Зокрема, інтерфейс SaveStrategy визначає єдиний метод save, а його конкретні реалізації – SaveAsTxt, SaveAsJson, SaveAsXml – забезпечують збереження файлів у різних форматах. Контекст FileSaver керує вибором стратегії та виконує збереження файлу, а клас File слугує моделлю даних, що зберігаються.

Відповіді на контрольні питання

1. Що таке шаблон проєктування?

   Формалізоване рішення типової задачі проєктування, яке показує взаємодію класів і об’єктів та містить рекомендації щодо застосування.

2. Навіщо використовувати шаблони проєктування?

   Для підвищення гнучкості, повторного використання коду, зрозумілості архітектури та спрощення підтримки системи.

3. Яке призначення шаблону «Стратегія»?

   Дозволяє змінювати поведінку об’єкта, вибираючи один з алгоритмів (стратегій), що досягають однієї мети різними способами.

4. Нарисуйте структуру шаблону «Стратегія»

   Контекст → Стратегія (інтерфейс) → Конкретні стратегії.

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

   - Контекст – виконує дії через стратегію.
   - Інтерфейс стратегії – визначає метод save.
   - Конкретні стратегії – реалізують алгоритм збереження. Контекст делегує роботу конкретній стратегії.

6. Яке призначення шаблону «Стан»?

   Дозволяє змінювати поведінку об’єкта залежно від його внутрішнього стану, ізолюючи логіку станів в окремі класи.

7. Нарисуйте структуру шаблону «Стан»

   Контекст → Інтерфейс стану → Конкретні стани.

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

   - Контекст – делегує виклики стану.
   - Інтерфейс стану– визначає методи поведінки.
   - Конкретні стани– реалізують конкретну поведінку.

9. Яке призначення шаблону «Ітератор»?

   Дозволяє послідовно обходити елементи колекції без розкриття її внутрішньої структури, виносячи логіку обходу з колекції.

10. Нарисуйте структуру шаблону «Ітератор»

    Колекція → Ітератор → Клієнт.

    Ітератор керує станом обходу та надає доступ до елементів.

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

    - Колекція – зберігає дані.

- Ітератор – відстежує стан обходу та надає елементи.
- Клієнт – використовує ітератор для перебору без знання внутрішньої структури.

12. В чому полягає ідея шаблону «Одинак»?

    Гарантує існування лише одного екземпляра класу та надає глобальну точку доступу до нього.

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

    Тому що він створює глобальні об’єкти зі станом, що ускладнює тестування, підтримку та порушує принцип єдиної відповідальності класу.

14. Яке призначення шаблону «Проксі»?

    Створює об’єкт-замінник для реального об’єкта, контролює доступ та додає додаткову логіку або оптимізацію, не змінюючи клієнтський код.

15. Нарисуйте структуру шаблону «Проксі»

    Клієнт → Проксі → Реальний об’єкт.

    Проксі реалізує той самий інтерфейс, що й реальний об’єкт, і делегує йому виклики.

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

    - Клієнт – взаємодіє через інтерфейс.
    - Інтерфейс – визначає методи.
    - Реальний об’єкт – виконує основну роботу.
    - Проксі – контролює доступ, кешує або об’єднує запити, делегуючи їх реальному об’єкту.