Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

з дисципліни «Теорія розробки прогрманого забезпечення»

Тема: «Вступ до паттернів проектування.»

Варіант «Powershell terminal»

Виконав:

студент групи ІА-32
Бечке Олексій Ігорович

Перевірив:

Мягкий Михайло Юрійович

Київ 2025

Зміст

Вступ

Тема: Вступ до паттернів проектування.

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

Тема проєкту: Powershell terminal (strategy, command, abstract factory, bridge, interpreter, client-server). Термінал для powershell повинен нагадувати типовий термінал з можливістю налаштування кольорів синтаксичних конструкцій, розміру вікна, фону вікна, а також виконання команд powershell і виконуваних файлів, а також працювати в декількох вікнах терміналу (у вкладках або одночасно шляхом розділення вікна).

Теоретичні відомості

Поняття шаблону проєктування

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях [5]. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову. Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої предметної області. Це є необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проєктування.

Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв'язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови.

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

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).

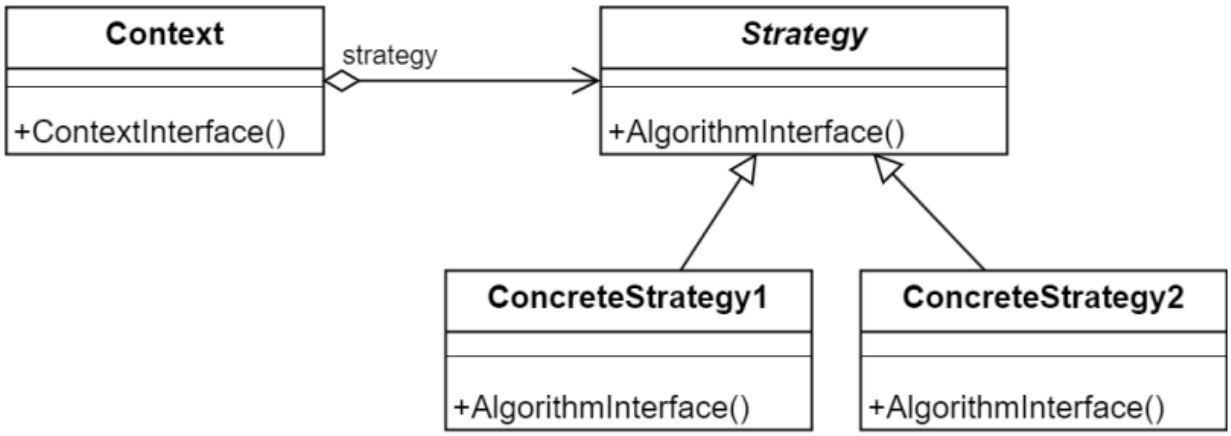


Рисунок 4.5. Структура патерну Стратегія

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.

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.

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

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

Хід роботи

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.

Реалізація шаблону проєктування для майбутньої системи

Ми використовуємо патерн Strategy використовується для гнучкого керування кольоровими темами інтерфейсу. Замість того, щоб прописувати жорсткі перевірки (на кшталт if-else) для кожного кольору безпосередньо в контролері, ми виносимо набори стилів (фон, текст, курсор) у окремі взаємозамінні класи-стратегії (DarkThemeStrategy, LightThemeStrategy), що дозволяє користувачеві миттєво перемикати вигляд терміналу «на льоту», а вам – легко додавати нові

теми в майбутньому, просто створюючи нові класи без втручання в основний код програми.

```java
package org.kpi.strategy;

/**
 * Інтерфейс Стратегії (Strategy).
 * Визначає алгоритм отримання кольорової схеми для веб-інтерфейсу.
 */
public interface WebThemeStrategy {  9 usages  2 implementations
    String getName(); // Назва теми для UI  no usages  2 implementations
    String getBackgroundColor();  no usages  2 implementations
    String getTextColor();  no usages  2 implementations
    String getPromptColor();  no usages  2 implementations
    String getCommandColor();  no usages  2 implementations
    String getErrorColor();  no usages  2 implementations
}
```

Рис. 1 – Код інтерфейсу WebThemeStrategy

```java
@Service  3 usages
public class ThemeContext {

    @Getter
    private WebThemeStrategy currentStrategy;
    private final Map<String, WebThemeStrategy> strategies;  3 usages

    public ThemeContext(Map<String, WebThemeStrategy> strategies) {
        this.strategies = strategies;
        this.currentStrategy = strategies.getOrDefault( key: "dark", new DarkThemeStrategy());
    }

    public void setStrategy(String strategyName) {  1 usage
        if (strategies.containsKey(strategyName)) {
            this.currentStrategy = strategies.get(strategyName);
        }
    }
}
```

Рис. 2 – Код класу ThemeContext

```java
@Component("dark")  1 usage
public class DarkThemeStrategy implements WebThemeStrategy {
    @Override  no usages
    public String getName() { return "Dark"; }

    @Override  no usages
    public String getBackgroundColor() { return "#1e1e1e"; }

    @Override  no usages
    public String getTextColor() { return "lightgray"; }

    @Override  no usages
    public String getPromptColor() { return "lightgreen"; }

    @Override  no usages
    public String getCommandColor() { return "yellow"; }

    @Override  no usages
    public String getErrorColor() { return "red"; }
}
```

Рис. 3 – Код класу DarkThemeStrategy

Зображення структури шаблону

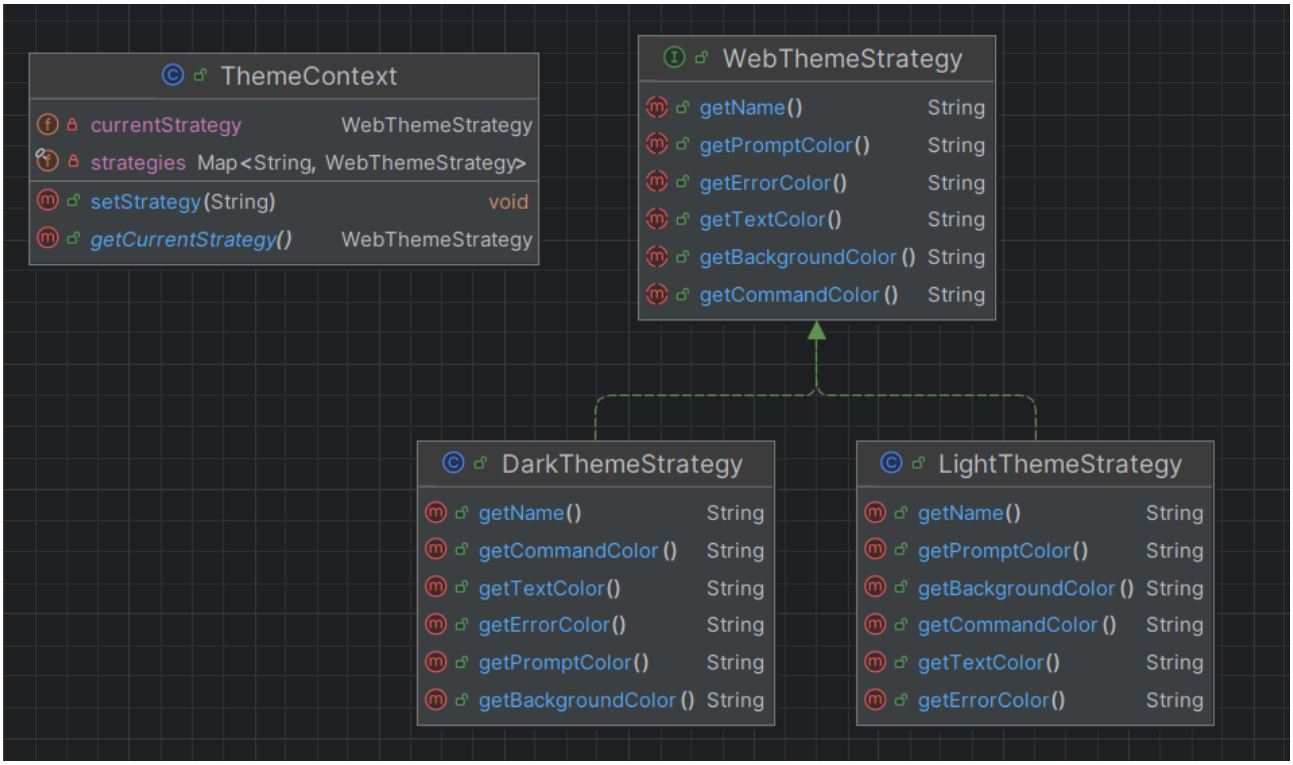


Рис. 4 – Структура шаблону Strategy

Структура реалізації:

- WebThemeStrategy – інтерфейс, що визначає методи для отримання кольорової схеми (фон, текст, курсор тощо).
- DarkThemeStrategy – реалізація темної теми (темний фон, світлий текст, відповідні кольори для синтаксису).
- LightThemeStrategy – реалізація світлої теми (світлий фон, темний текст, адаптовані кольори для синтаксису).
- ThemeContext – контекст, що зберігає посилання на поточну стратегію та дозволяє динамічно підміняти її (перемикати тему) під час виконання.

Такий підхід дозволяє відокремити візуальне оформлення від логіки відображення, завдяки чому можна легко додавати нові теми або змінювати існуючі без втручання в основний код контролерів та HTML-шаблонів.

Питання до лабораторної роботи

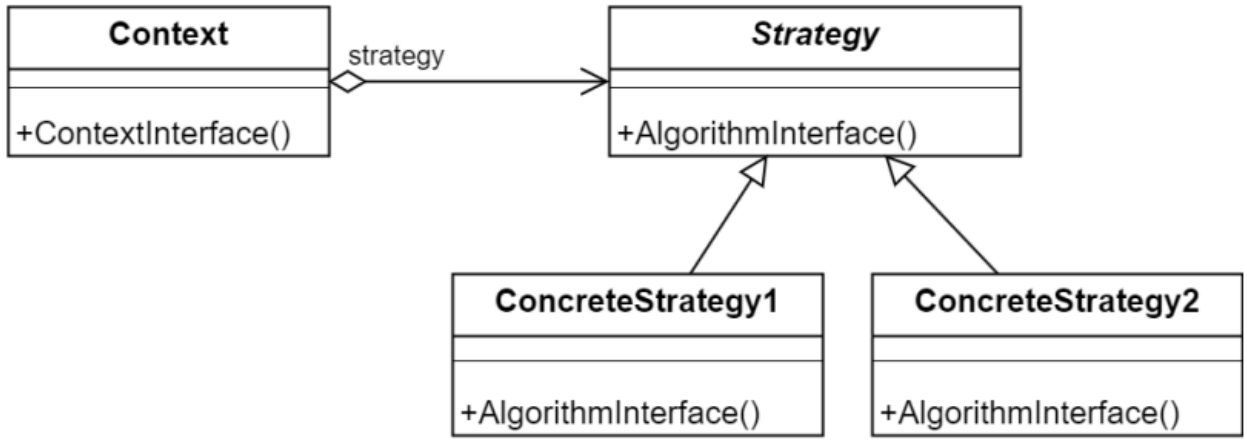
1. Що таке шаблон проєктування?
   Шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах

2. Навіщо використовувати шаблони проєктування?

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи.
Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

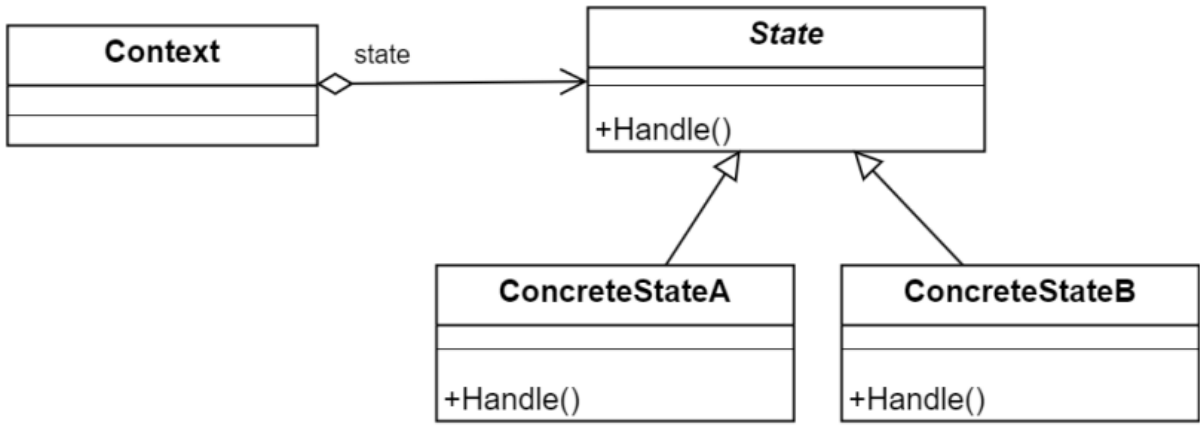
3. Яке призначення шаблону «Стратегія»?
   Шаблон «Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

4. Нарисуйте структуру шаблону «Стратегія».



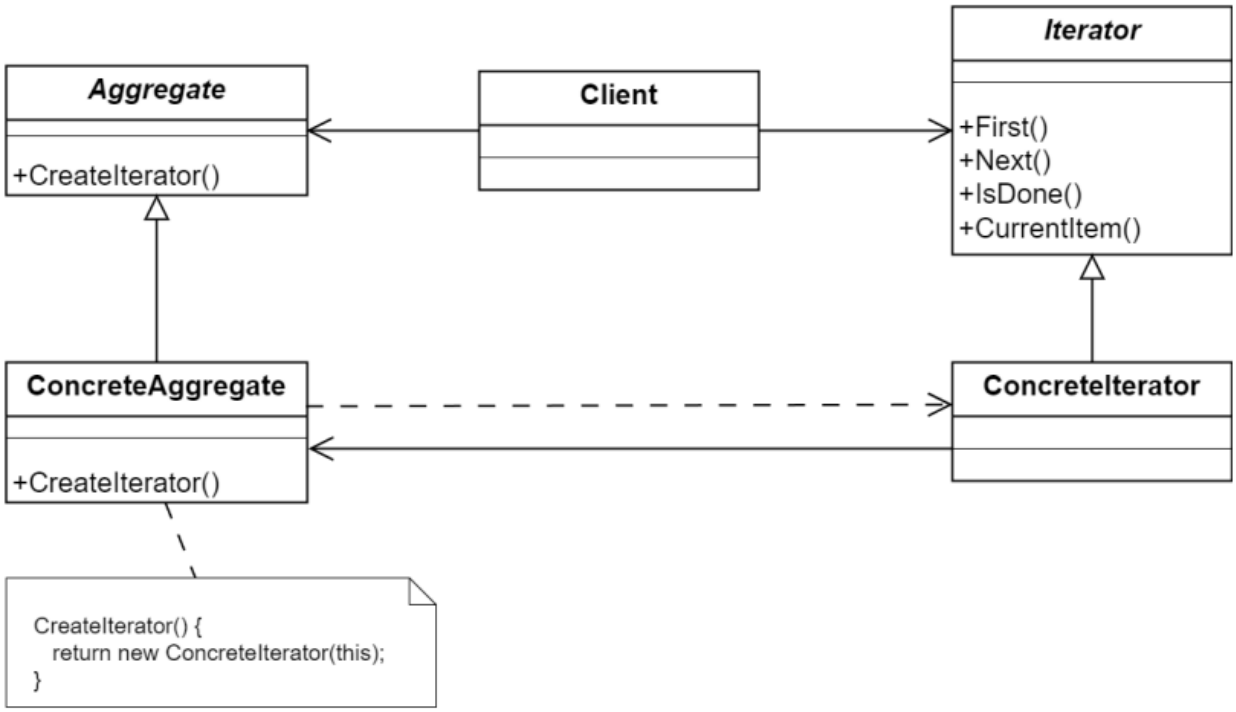
5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?
   - Context (Контекст): Клас, який використовує стратегію. Він містить посилання на об'єкт типу Strategy та спілкується з ним через спільний інтерфейс, не знаючи конкретного класу реалізації.
   - Strategy (Стратегія): Загальний інтерфейс (або абстрактний клас) для всіх підтримуваних алгоритмів. Він визначає метод (наприклад, AlgorithmInterface()), який має бути реалізований у конкретних стратегіях.
   - ConcreteStrategy (Конкретна Стратегія): Різні класи (наприклад, ConcreteStrategy1, ConcreteStrategy2), що реалізують інтерфейс Strategy. Кожен із них містить власну реалізацію алгоритму.
6. Яке призначення шаблону «Стан»?
   Шаблон «State» (Стан) дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?
    - State (Стан): Загальний інтерфейс, у який виносяться поля, властивості, методи та дії, пов'язані зі станом.
    - ConcreteState (Конкретний стан, наприклад, ConcreteStateA, ConcreteStateB): Окремі класи, що реалізують загальний інтерфейс State. Кожен такий клас представляє конкретний стан і визначає свою унікальну поведінку.
    - Context (Контекст): Клас, об'єкти якого мають змінювати свою поведінку залежно від внутрішнього стану. Він містить посилання на об'єкт типу State.
9. Яке призначення шаблону «Ітератор»?
    «Iterator» (Ітератор) являє собою шаблон реалізації об'єкта доступу до набору (колекції, агрегату) елементів без розкриття внутрішніх механізмів реалізації. Ітератор виносить функціональність перебору колекції елементів з самої колекції, таким чином досягається розподіл обов'язків: колекція відповідає за зберігання даних, ітератор – за прохід по колекції

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?
    - Iterator (Ітератор): Інтерфейс, який визначає стандартні методи для доступу до елементів та переміщення по них. Зазвичай це методи First() (початок), Next() (наступний), IsDone() (перевірка завершення) та CurrentItem() (поточний елемент) .
    - ConcreteIterator (Конкретний ітератор): Клас, що реалізує інтерфейс Iterator. Він відповідає за відстеження поточної позиції при обході колекції та реалізує конкретний алгоритм переміщення.
    - Aggregate (Агрегат): Інтерфейс, який визначає метод для створення об'єкта-ітератора (наприклад, CreateIterator()) .
    - ConcreteAggregate (Конкретний агрегат): Клас конкретної колекції, який реалізує інтерфейс Aggregate. Він зберігає дані та повертає екземпляр відповідного ConcreteIterator, який вміє працювати з цією колекцією .
    - Client (Клієнт): Використовує ітератор та агрегат через їхні інтерфейси для доступу до елементів колекції, не знаючи про її внутрішню структуру.
12. В чому полягає ідея шаблона «Одинак»?

«Singleton» (Одинак) являє собою клас в термінах ООП, який може мати не більше одного об'єкта. Насправді, кількість об'єктів можна задати (тобто не можна створити більш n об'єктів даного класу). Даний об'єкт найчастіше зберігається як статичне поле в самому класі.

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

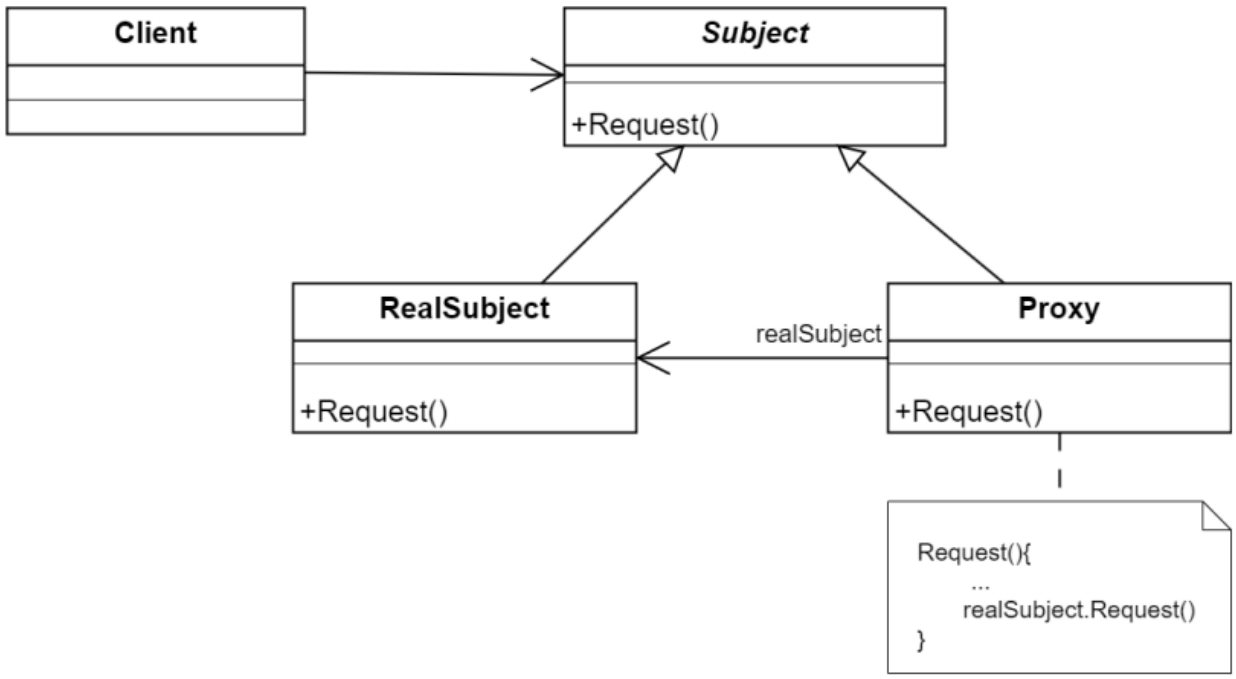
    Це пов'язано з тим, що «одинаки» представляють собою глобальні дані (як глобальна змінна), що мають стан. Стан глобальних об'єктів важко відслідковувати і підтримувати коректно; також глобальні об'єкти важко

тестуються і вносять складність в програмний код (у всіх ділянках коду виклик в одне єдине місце з «одинаком»; при зміні підходу доведеться змінювати масу коду).

14. Яке призначення шаблону «Проксі»?

    «Proxy» (Проксі) – об'єкти є об'єктами-заглушками або двійниками/замінниками для об'єктів конкретного типу. Зазвичай, проксі об'єкти вносять додатковий функціонал або спрощують взаємодію з реальними об'єктами.

15. Нарисуйте структуру шаблону «Проксі».



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

- Subject (Суб'єкт): Загальний інтерфейс, який визначає методи, спільні для реального об'єкта та його замісника. Завдяки цьому проксі може використовуватися скрізь, де очікується реальний об'єкт.
- RealSubject (Реальний суб'єкт): Клас, що представляє реальний об'єкт, який виконує основну роботу.
- Proxy (Замісник): Клас, який зберігає посилання на реальний суб'єкт (RealSubject) і реалізує інтерфейс Subject. Він виступає посередником, контролюючи доступ до реального об'єкта.
- Client (Клієнт): Взаємодіє з об'єктами через інтерфейс Subject, не знаючи, чи працює він з реальним об'єктом, чи з його замісником.

Висновок

Під час виконання лабораторної роботи я вивчив структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи. Реалізував шаблон «Strategy» у проєкті Power Shell термінал.