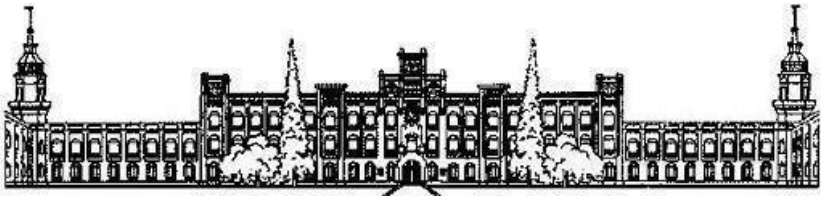


Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

з дисципліни «Технології розроблення
програмного забезпечення»

Тема: «Вступ до паттернів проектування.»

Виконав:
студент групи ІА-32
Діденко Я.О

Перевірив:
Мягкий М.Ю

Київ 2025

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

18. **Shell (total commander)** (state, prototype, factory method, template method, interpreter, client-server)

Оболонка повинна вміти виконувати основні дії в системі – перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

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. 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 «State»

Призначення: Шаблон «State» (Стан) дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану . Наприклад, відсоток нарахованих на картковий рахунок грошей залежить від стану картки: Visa Electron, Classic, Platinum і т.д. Або обсяг послуг, які надані хостинг компанією, змінюється в залежності від обраного тарифного плану (стану членства – бронзовий, срібний або золотий клієнт). Реалізація даного шаблону полягає в наступному: пов'язані зі станом поля, властивості, методи і дії виносяться в окремий загальний інтерфейс (State); кожен стан являє собою окремий клас (ConcreteStateA, ConcreteStateB), які реалізують загальний інтерфейс.

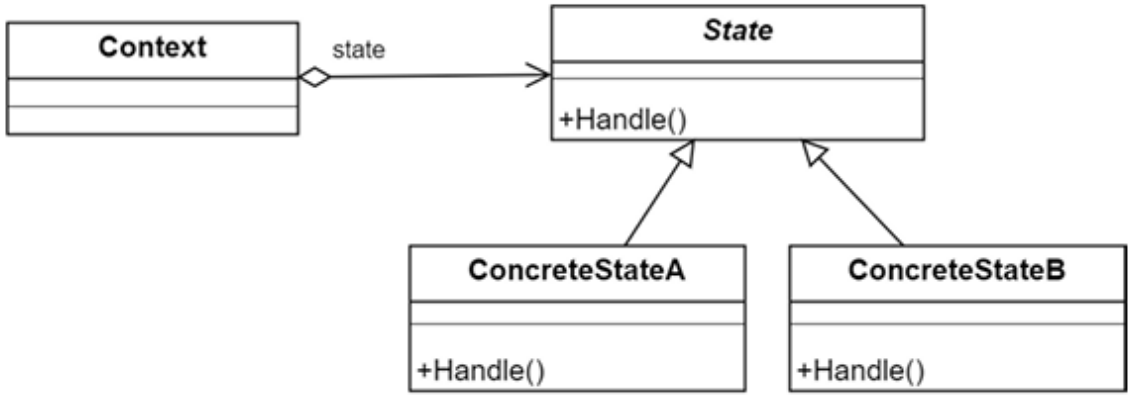


рис 1. Структура патерну State

Об’єкти, що мають стан (Context), при зміні стану просто записують новий об'єкт в поле state, що призводить до повної зміни поведінки об'єкта.
Це дозволяє легко додавати в майбутньому і обробляти нові стани, відокремлювати залежні від стану елементи об'єкта в інших об'єктах, і відкрито проводити заміну стану (що має сенс у багатьох випадках).

Проблема: Ви розробляєте систему, яка складається з мобільних клієнтів та центрального сервера. Для отримання запитів від клієнтів ви створюєте модуль Listener. Але вам потрібно щоб під час старту, поки сервер не запустився Listener не приймав запити від клієнтів, а після команди shutdown, поки сервер зупиняється, Listener вже не приймав запити від клієнтів, але відповіді, якщо вони будуть готові, відправив.

Рішення: Тут явно видно три стани для Listener з різною поведінкою: initializing, open, closing. Тому для вирішення краще за все підходить патерн State.

Визначаємо загальний інтерфейс IListenerState з методами, які по різному працюють в різних станах. Під кожний стан створюємо клас з реалізацією цього інтерфейсе. Контекст Listener при цьому всі виклики буде перенаправляти на об'єкт стану простим делегуванням. При старті він (Listener) буде посилатися на об'єкт стану InitializingState, знаходячись в якому Listener, фактично, буде ігнорувати всі вхідні запити. Після того як система запуститься і завантажить всі базові дані для роботи, Listener буде переключено в робочий стан, наприклад, викликом методу Open(). Після цього Listener буде посилатися на об'єкт OpenState і буде відпрацьовувати всі вхідні повідомлення в звичайному режимі. При запуску виключення системи, Listener буде переключено в стан ClosingState, викликом методу Close().

Переваги:
+ Код специфічний для окремого стану реалізується в класі стану.
+ Класи та об’єкти станів можна використовувати з різними контекстами, за рахунок чого збільшується гнучкість системи.
+ Код контексту простіше читати, тому що вся залежна від станів логіка винесена в інші класи.
+ Відносно легко додавати нові стани, головне правильно змінити переходи між станами.

Недолік:
- Клас контекст стає складніше через ускладнений механізм переключення станів.

Хід роботи

1. Загальний опис виконаної роботи

В рамках даної лабораторної роботи було реалізовано шаблон проектування «Стан» (State).

Основна мета полягала в тому, щоб продемонструвати, як об'єкт (FileManager) може змінювати свою поведінку залежно від свого внутрішнього стану (наприклад, стан очікування, стан копіювання, стан перегляду тощо), уникаючи при цьому використання громіздких умовних конструкцій (if-else або switch).

Програмна система складається з наступних ключових компонентів:

- **Контекст (FileManager):** Основний клас, з яким взаємодіє користувач. Він делегує виконання всіх операцій своєму поточному стану.
- **Інтерфейс стану (State):** Загальний інтерфейс, що визначає набір методів, які мають реалізувати всі конкретні стани.
- **Конкретні стани:** Класи, що реалізують інтерфейс State (IdleState, ViewingState, CopyingState, DeletingState, MovingState). Кожен клас інкапсулює логіку поведінки для одного конкретного стану та відповідає за перехід до наступного стану.

Поведінка файлового менеджера змінюється не через зміну коду в самому класі FileManager, а шляхом заміни об'єкта-стану, на який він посилається.

2. Опис класів та інтерфейсів програмної системи.

Клас Main.java

Цей клас є точкою входу в програму. Його єдина мета - продемонструвати роботу патерну. Тут створюється екземпляр FileManager та послідовно викликаються його методи, щоб показати, як змінюється поведінка системи та її вивід в консоль залежно від послідовності дій та переходів між станами.

```java
public class Main {
    public static void main(String[] args) {
        FileManager commander = new FileManager();

        System.out.println("=== Total Commander Demo ===\n");

        commander.viewFiles();
        commander.copyFile( fileName: "document.txt");
        commander.deleteFile( fileName: "temp.tmp");
        commander.moveFile( fileName: "photo.jpg");
        commander.viewFiles();

        System.out.println("\n=== Повторні операції ===\n");

        commander.copyFile( fileName: "video.mp4");
        commander.copyFile( fileName: "music.mp3");
        commander.viewFiles();
    }
}
```

рис 2.1 - код класу Main.java

```
=== Total Commander Demo ===

[Idle] Переглядаємо файли в поточній директорії
[Viewing] Завершуємо перегляд та копіюємо файл: document.txt
[Copying] Копіювання завершено. Видаляємо файл: temp.tmp
[Deleting] Видалення завершено. Переміщуємо файл: photo.jpg
[Moving] Переміщення завершено. Переглядаємо файли

=== Повторні операції ===

[Viewing] Завершуємо перегляд та копіюємо файл: video.mp4
[Copying] Копіюємо ще один файл: music.mp3
[Copying] Копіювання завершено. Переглядаємо файли
```

рис 2.2 - результати запуску програми

Клас FileManager.java (Контекст)

Це ключовий клас системи, поведінка якого динамічно змінюється. Він зберігає посилання на свій поточний стан у приватній змінній currentState. Всі публічні методи (viewFiles, copyFile тощо) не містять власної логіки, а просто делегують виклик відповідному методу поточного об'єкта-стану. Клас також надає публічний метод setState(), який дозволяє об'єктам-станам змінювати поточний стан менеджера, реалізуючи таким чином переходи між станами.

```
1   public class FileManager {  26 usages
2       private State currentState;  6 usages
3
4       public FileManager() {  1 usage
5           this.currentState = new IdleState();
6       }
7
8       public void setState(State state) {  16 usages
9           this.currentState = state;
10      }
11
12      public void viewFiles() {  3 usages
13          currentState.viewFiles( context: this);
14      }
15
16      public void copyFile(String fileName) {  3 usages
17          currentState.copyFile( context: this, fileName);
18      }
19
20      public void deleteFile(String fileName) {  1 usage
21          currentState.deleteFile( context: this, fileName);
22      }
23
24      public void moveFile(String fileName) {  1 usage
25          currentState.moveFile( context: this, fileName);
26      }
27  }
```

рис 2.3 - код класу FileManager.java

Інтерфейс State.java

Це базовий інтерфейс, який визначає "контракт" для всіх можливих станів файлового менеджера. Він оголошує набір методів, що відповідають основним діям системи. Важливо, що кожен метод приймає посилання на об'єкт контексту (FileManager), що дозволяє будь-якому стану викликати публічні методи контексту, зокрема setState(), для здійснення переходу в новий стан.

```
1  public interface State {  8 usages  5 implementations
2      void viewFiles(FileManager context);  1 usage  5 implementations
3      void copyFile(FileManager context, String fileName);  1 usage  5 implementations
4      void deleteFile(FileManager context, String fileName);  1 usage  5 implementations
5      void moveFile(FileManager context, String fileName);  1 usage  5 implementations
6  }
```

рис 2.4 - код інтерфейсу State.java

Класи конкретних станів (IdleState, CopyingState та інші)

Ці класи є ядром патерну, оскільки саме в них інкапсульована вся специфічна для кожного стану логіка. Кожен такий клас реалізує інтерфейс State і надає свою версію поведінки для кожної з операцій.

Ключовим аспектом їхньої роботи є керування переходами. Після виконання своєї дії (наприклад, ініціації копіювання), клас-стан викликає метод context.setState(), передаючи в нього екземпляр нового стану, в який система має перейти. Це дозволяє створювати ланцюжки логічних переходів між станами.

Для демонстрації наведемо приклад реалізації початкового стану IdleState.

Приклад: IdleState.java

Цей клас представляє стан очікування, в якому файловий менеджер перебуває на початку роботи або після завершення попередньої операції. З цього стану можна ініціювати будь-яку дію (перегляд, копіювання тощо). Після виклику будь-якого методу, IdleState виводить повідомлення про

початок операції та негайно переводить контекст (FileManager) у відповідний новий стан (наприклад, ViewingState або CopyingState).

```java
public class IdleState implements State {  1 usage
    @Override  1 usage
    public void viewFiles(FileManager context) {
        System.out.println("[Idle] Переглядаємо файли в поточній директорії");
        context.setState(new ViewingState());
    }

    @Override  1 usage
    public void copyFile(FileManager context, String fileName) {
        System.out.println("[Idle] Копіюємо файл: " + fileName);
        context.setState(new CopyingState());
    }

    @Override  1 usage
    public void deleteFile(FileManager context, String fileName) {
        System.out.println("[Idle] Видаляємо файл: " + fileName);
        context.setState(new DeletingState());
    }

    @Override  1 usage
    public void moveFile(FileManager context, String fileName) {
        System.out.println("[Idle] Переміщуємо файл: " + fileName);
        context.setState(new MovingState());
    }
}
```

рис 2.5 - код класу IdleState.java

Примітка: Інші класи станів (ViewingState, CopyingState, DeletingState, MovingState) реалізовані за аналогічним принципом, визначаючи унікальну поведінку та правила переходу для кожної конкретної ситуації.

3. Діаграма класів

Для візуалізації архітектури розробленої системи та взаємозв'язків між її компонентами була побудована UML-діаграма класів. Вона наочно демонструє структуру, що відповідає шаблону проектування «Стан».

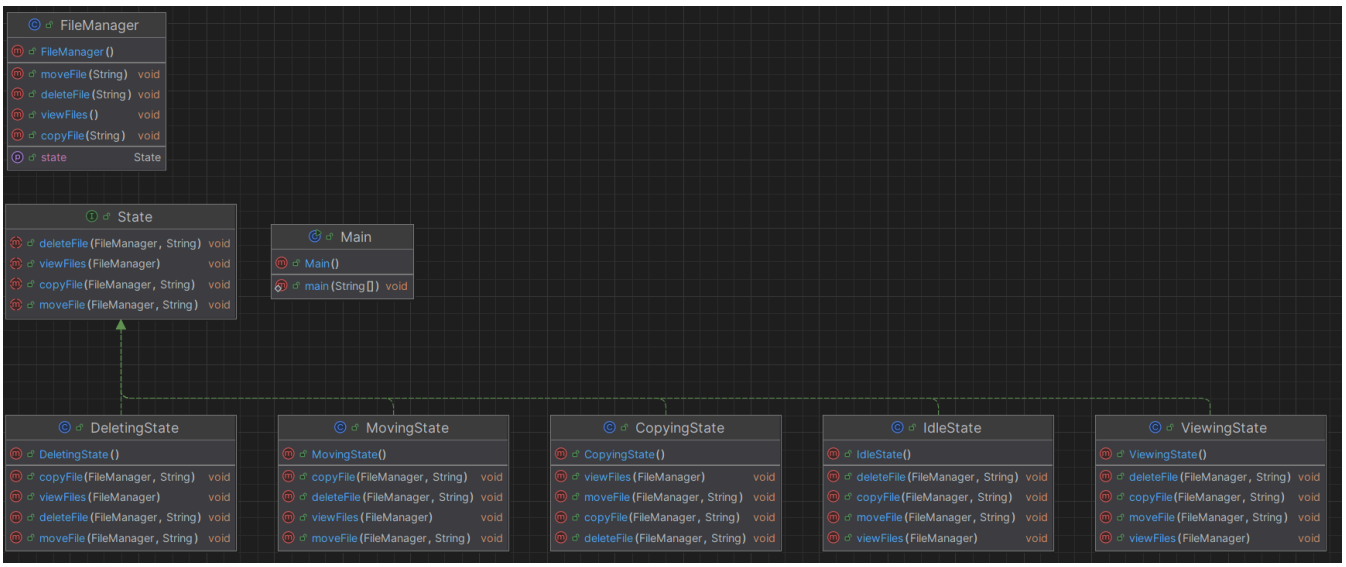


рис 3.1 - загальний вигляд діаграми класів

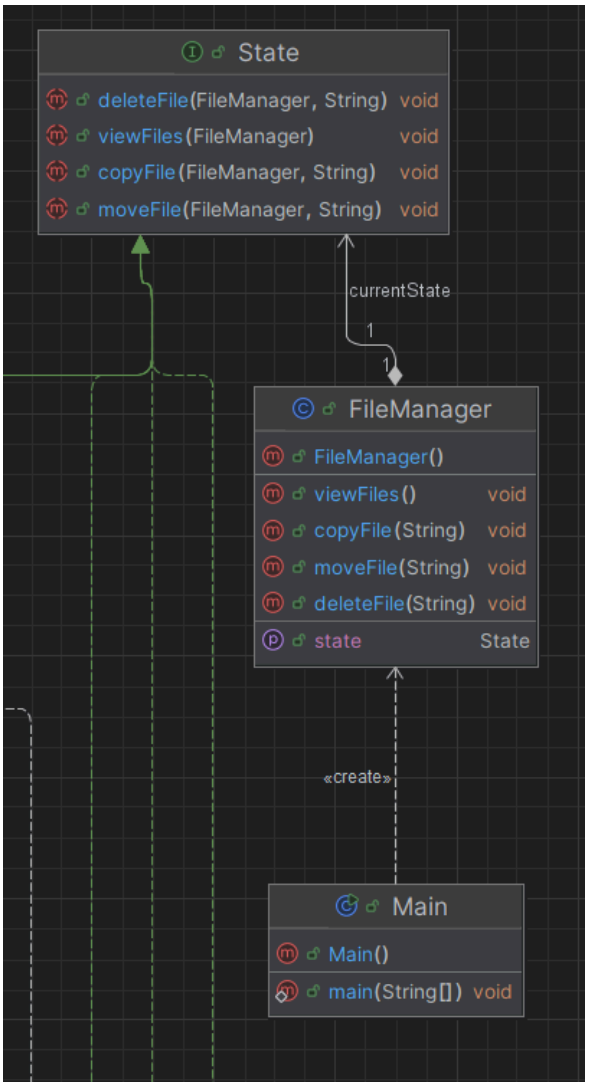


рис 3.2 - вигляд діаграми з параметром Show Dependencies

На діаграмі чітко видно ключові елементи патерну:

- **Асоціація** між FileManager (Контекст) та State (Стан), яка показує, що контекст має посилання на об'єкт поточного стану.
- **Реалізація** (успадкування від інтерфейсу) між конкретними станами (IdleState, CopyingState та іншими) та інтерфейсом State, що зобов'язує їх реалізовувати спільний набір методів.
- **Залежність** класу Main від FileManager, оскільки Main створює та використовує об'єкт менеджера для демонстрації роботи.

Діаграма візуально підтверджує, що архітектура системи побудована відповідно до канонічної структури шаблону «Стан».

4. Висновки.

Висновки: в ході виконання лабораторної роботи було успішно реалізовано та досліджено шаблон проектування «Стан», продемонструвавши на практичному прикладі файлового менеджера, як даний патерн дозволяє об'єкту динамічно змінювати свою поведінку залежно від внутрішнього стану. Було підтверджено ключові переваги цього підходу: вся логіка, що стосується конкретного стану, інкапсулюється в окремому класі, що відповідає принципу єдиної відповідальності та робить код чистішим; клас-контекст (FileManager) позбавляється складної умовної логіки, що значно спрощує його для розуміння та підтримки; система стає гнучкою та розширюваною, оскільки додавання нових станів не вимагає модифікації існуючого коду. Таким чином, доведено, що шаблон «Стан» є потужним та ефективним інструментом для проектування систем, в яких об'єкти мають складну, залежну від стану поведінку, та дозволяє створювати керовані і добре структуровані програмні рішення.

Контрольні запитання

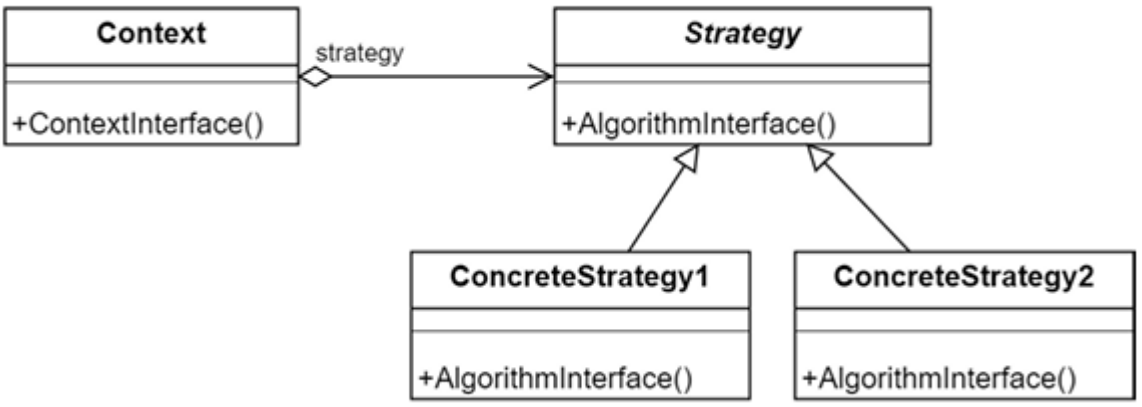
1. Що таке шаблон проєктування?
Шаблон проєктування (design pattern) - це перевірене, універсальне та повторно використовуване рішення для типової проблеми, що часто виникає в рамках проєктування програмного забезпечення. Це не готовий фрагмент коду, а скоріше концепція або архітектурний ескіз, який описує взаємодію об'єктів та класів для вирішення задачі.

2. Навіщо використовувати шаблони проєктування?
Використання шаблонів проєктування надає декілька переваг:

- **Перевірені рішення:** Вони пропонують рішення, які вже довели свою ефективність на практиці.
- **Спільний словник:** Вони створюють єдину мову для розробників, що спрощує обговорення архітектури та комунікацію в команді.
- **Підвищення якості коду:** Код стає більш гнучким, розширюваним, читабельним та легким для підтримки.
- **Прискорення розробки:** Не потрібно "винаходити велосипед" для вирішення стандартних проблем.

3. Яке призначення шаблону «Стратегія»?
Призначення шаблону «Стратегія» (Strategy) - визначити сімейство алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Це дозволяє клієнту вибирати потрібний алгоритм під час виконання програми, не змінюючи код самого клієнта.

4. Нарисуйте структуру шаблону «Стратегія».



1. Існує клас **Context**, який містить посилання (асоціацію) на інтерфейс **Strategy**.
2. Існує інтерфейс **Strategy**, який оголошує один загальний метод (наприклад, execute()).
3. Існує кілька класів **ConcreteStrategyA**, **ConcreteStrategyB** і т.д., які реалізують інтерфейс **Strategy**. Кожен з них надає свою унікальну реалізацію методу execute().

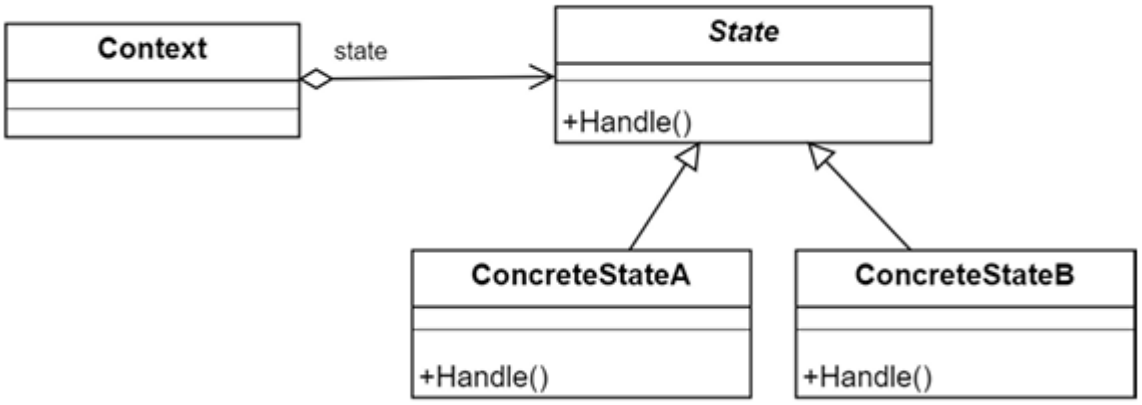
5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

- **Context (Контекст):** Клас, поведінка якого повинна змінюватися. Він зберігає посилання на один з об'єктів-стратегій. Він не реалізує алгоритм сам, а делегує його виконання об'єкту-стратегії.
- **Strategy (Стратегія):** Загальний інтерфейс для всіх конкретних стратегій. Він визначає метод, який повинні реалізувати всі алгоритми.
- **ConcreteStrategy (Конкретна стратегія):** Клас, що реалізує конкретний алгоритм, слідуючи інтерфейсу Strategy.

Взаємодія: Клієнт створює об'єкт Context та один з об'єктів ConcreteStrategy. Потім він передає стратегію контексту. Коли клієнт викликає метод у контексту, той, у свою чергу, викликає відповідний метод у об'єкта-стратегії, який йому передали.

6. Яке призначення шаблону «Стан»?
Призначення шаблону «Стан» (State) - дозволити об'єкту змінювати свою поведінку, коли змінюється його внутрішній стан. Зовні це виглядає так, ніби об'єкт змінив свій клас.

7. Нарисуйте структуру шаблону «Стан».



1. Існує клас **Context**, який містить посилання (асоціацію) на інтерфейс **State**.
2. Існує інтерфейс **State**, який оголошує методи, що відповідають різним варіантам поведінки.

3. Існує кілька класів **ConcreteStateA**, **ConcreteStateB** і т.д., які реалізують інтерфейс **State**. Вони містять логіку поведінки для конкретного стану і можуть ініціювати перехід до іншого стану.

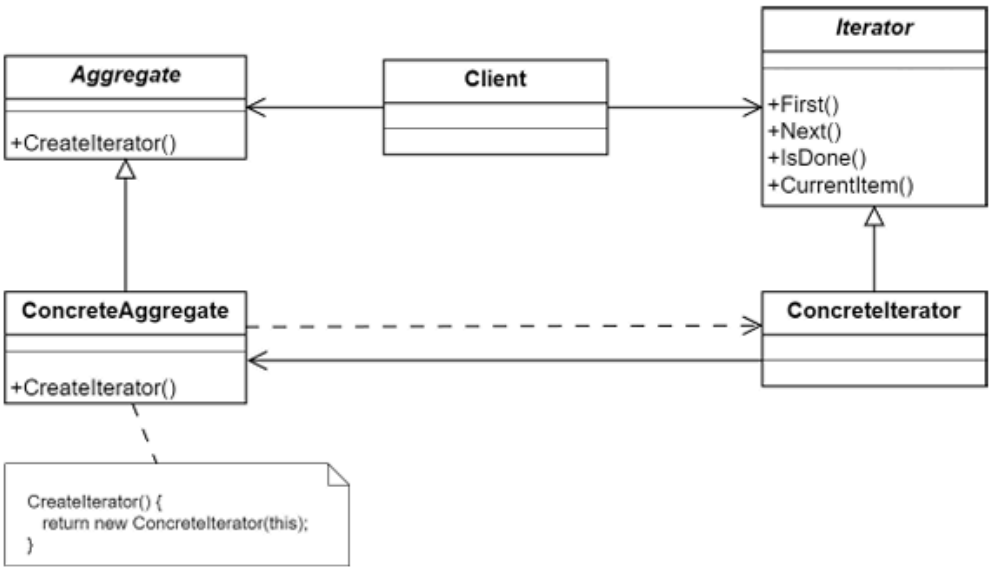
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

- **Context (Контекст):** Клас, поведінка якого змінюється. Зберігає посилання на об'єкт поточного стану і делегує йому виконання запитів.
- **State (Стан):** Загальний інтерфейс для всіх станів. Визначає набір методів, які реалізують поведінку.
- **ConcreteState (Конкретний стан):** Клас, що реалізує поведінку, пов'язану з конкретним станом контексту. Він також відповідає за перемикання контексту в новий стан.

Взаємодія: Клієнт взаємодіє з об'єктом Context. Контекст делегує виконання запиту своєму поточному об'єкту-стану. Об'єкт-стан виконує дію і може змінити поточний стан контексту, передавши йому новий об'єкт-стан.

9. Яке призначення шаблону «Ітератор»?
Призначення шаблону «Ітератор» (Iterator) - надати стандартизований спосіб послідовного доступу до елементів колекції (наприклад, списку, дерева, масиву), не розкриваючи її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



1. Існує інтерфейс **Iterator** з методами, як-от hasNext() та next().
2. Існує інтерфейс **Aggregate** (або Collection) з методом createIterator().
3. Існує клас **ConcreteIterator**, який реалізує інтерфейс Iterator для конкретної колекції.
4. Існує клас **ConcreteAggregate**, який реалізує інтерфейс Aggregate і повертає екземпляр ConcreteIterator.

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

- **Iterator:** Інтерфейс, що визначає методи для обходу колекції.
- **ConcreteIterator:** Клас, що реалізує інтерфейс Iterator і відстежує поточну позицію в колекції.
- **Aggregate:** Інтерфейс, що визначає метод для створення об'єкта-ітератора.
- **ConcreteAggregate:** Клас колекції, що реалізує Aggregate і створює конкретний ітератор для себе.

Взаємодія: Клієнт отримує об'єкт-ітератор від об'єкта-колекції. Після цього клієнт працює тільки з ітератором, викликаючи його методи для перебору елементів, і не залежить від внутрішньої реалізації колекції.

12. В чому полягає ідея шаблона «Одинак»?
Ідея шаблону «Одинак» (Singleton) полягає в тому, щоб гарантувати, що для певного класу буде створено лише один екземпляр, і надати глобальну точку доступу до цього екземпляра.

13. Чому шаблон «Одинак» вважають «анти-шаблоном»?
Його часто вважають «анти-шаблоном» з кількох причин:

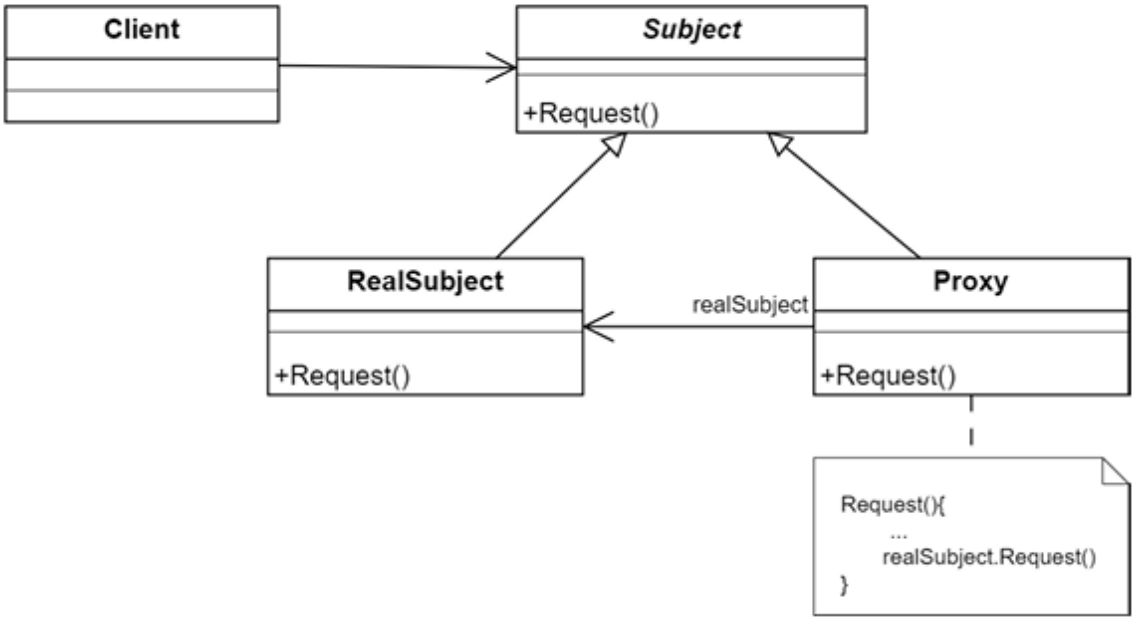
- **Порушує принцип єдиної відповідальності:** Клас починає відповідати не тільки за свою основну логіку, але й за контроль над кількістю своїх екземплярів.
- **Створює глобальний стан:** Глобальні змінні ускладнюють відстеження залежностей і роблять код менш передбачуваним.
- **Ускладнює тестування:** Важко ізолювати та тестувати класи, які залежать від одинака, оскільки неможливо легко замінити його на тестовий об'єкт (мок).

- **Приховує залежності:** Замість того, щоб передавати залежності через конструктор, клас може "тихо" отримати доступ до одинака, що робить архітектуру неочевидною.

14. Яке призначення шаблону «Проксі»?

Призначення шаблону «Проксі» (Proxy) - надати об'єкт-замісник (сурогат) для іншого об'єкта, щоб контролювати доступ до нього. Проксі може додавати додаткову логіку: кешування, перевірку прав доступу, логування, лініву ініціалізацію тощо.

15. Нарисуйте структуру шаблону «Проксі».



1. Існує спільний інтерфейс **Subject**, який визначає методи, доступні як для реального об'єкта, так і для проксі.
2. Існує клас **RealSubject**, який реалізує інтерфейс Subject і містить основну бізнес-логіку.
3. Існує клас **Proxy**, який також реалізує інтерфейс Subject і містить посилання на об'єкт RealSubject.

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

- **Subject:** Спільний інтерфейс, який дозволяє клієнту працювати з проксі так само, як і з реальним об'єктом.
- **RealSubject:** Справжній об'єкт, роботу якого проксі представляє.

- **Proxy:** Об'єкт-замісник. Він перехоплює виклики до RealSubject, виконує свою додаткову логіку і потім (за необхідності) делегує виклик реальному об'єкту.

Взаємодія: Клієнт працює з об'єктом Proxy через інтерфейс Subject. Для клієнта немає різниці, з ким він спілкується. Proxy отримує запит, виконує додаткові дії (наприклад, перевіряє кеш або права доступу) і, якщо потрібно, передає запит об'єкту RealSubject, який він контролює.