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 № 6

із дисципліни ««Технології розроблення програмного забезпечення»
Тема: «Патерни проектування»

Виконав:
студент групи ІА-31:
Кунда А.П.

Перевірив:
Мягкий М.Ю

Київ 2025

Тема: Патерни проектування

Тема проєкту: 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.). Autosave\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.

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом:
https://github.com/Octopus663/json-tool

Хід роботи

1) Ознайомитись з короткими теоретичними відомостями

Шаблон «Observer»

Призначення: Шаблон визначає залежність «один-до-багатьох» таким чином, що коли один об'єкт змінює власний стан, усі інші об'єкти отримують про це сповіщення і мають можливість змінити власний стан також [6].

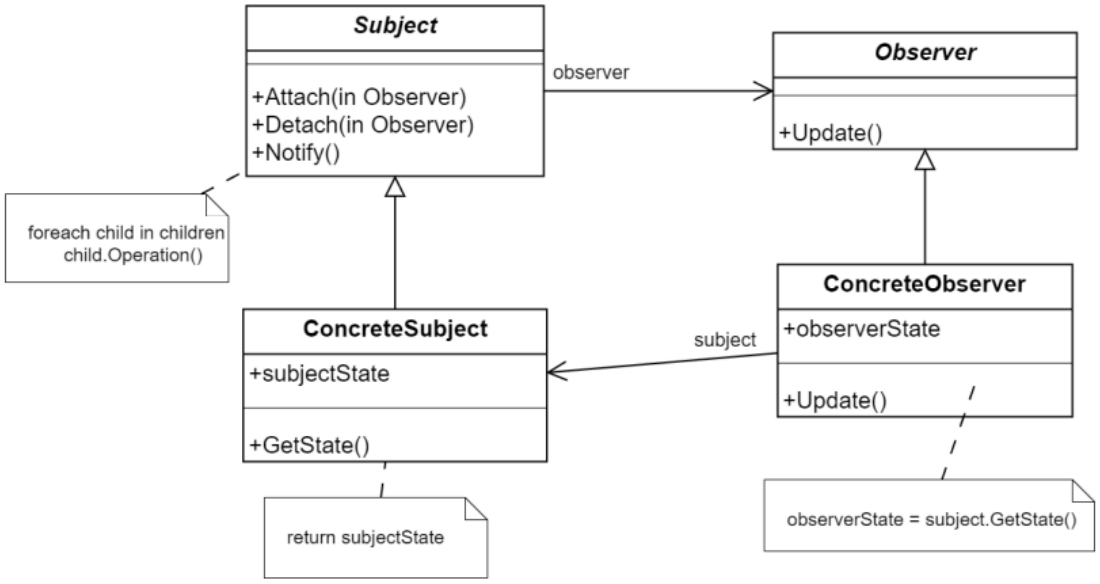


Рисунок 6.4. Структура патерна «Спостерігач»

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

Для реалізації даного патерну нам потрібно два класи(інтерфейси):

1) Observer.java

```
package com.example.jsontool.observer;

public interface Observer {  no usages  new *
    void update(Subject subject);  no usages  new *
}
```

Інтерфейс спостерігача визначає метод, який Суб'єкт буде викликати при зміні стану.

2) Subject java

```
package com.example.jsontool.observer;

public interface Subject {  1 usage  new *
    void addObserver(Observer observer);  no usages  new *
    void removeObserver(Observer observer);  no usages  new *
    void notifyObservers();  no usages  new *
}
```

Інтерфейс суб'єкта визначає методи для керування підписниками.

3)Реалізувати один з розглянутих шаблонів за обраною темою.

4) Реалізувати не менше 3-х класів відповідно до обраної теми.

Створимо клас конкретного суб’єкта Це буде наш "редактор", який зберігає поточний текст JSON і має список підписників.

```java
import com.example.jsontool.observer.Observer;
import com.example.jsontool.observer.Subject;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service  new *
public class JsonEditorService implements Subject {

    private final List<Observer> observers = new ArrayList<>();  3 usages

    private String jsonContent;  2 usages
    private String schemaContent;  2 usages

    @Override  no usages  new *
    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    @Override  no usages  new *
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    @Override  1 usage  new *
    public void notifyObservers() {
        // Повідомити кожного підписника
        for (Observer observer : observers) {
            observer.update( subject: this);
        }
    }

    public void setTextContents(String jsonContent, String schemaContent) {  no usages  new *
        this.jsonContent = jsonContent;
        this.schemaContent = schemaContent;

        notifyObservers();
    }

    public String getJsonContent() {  no usages  new *
        return jsonContent;
    }
```

JsonEditorService.java

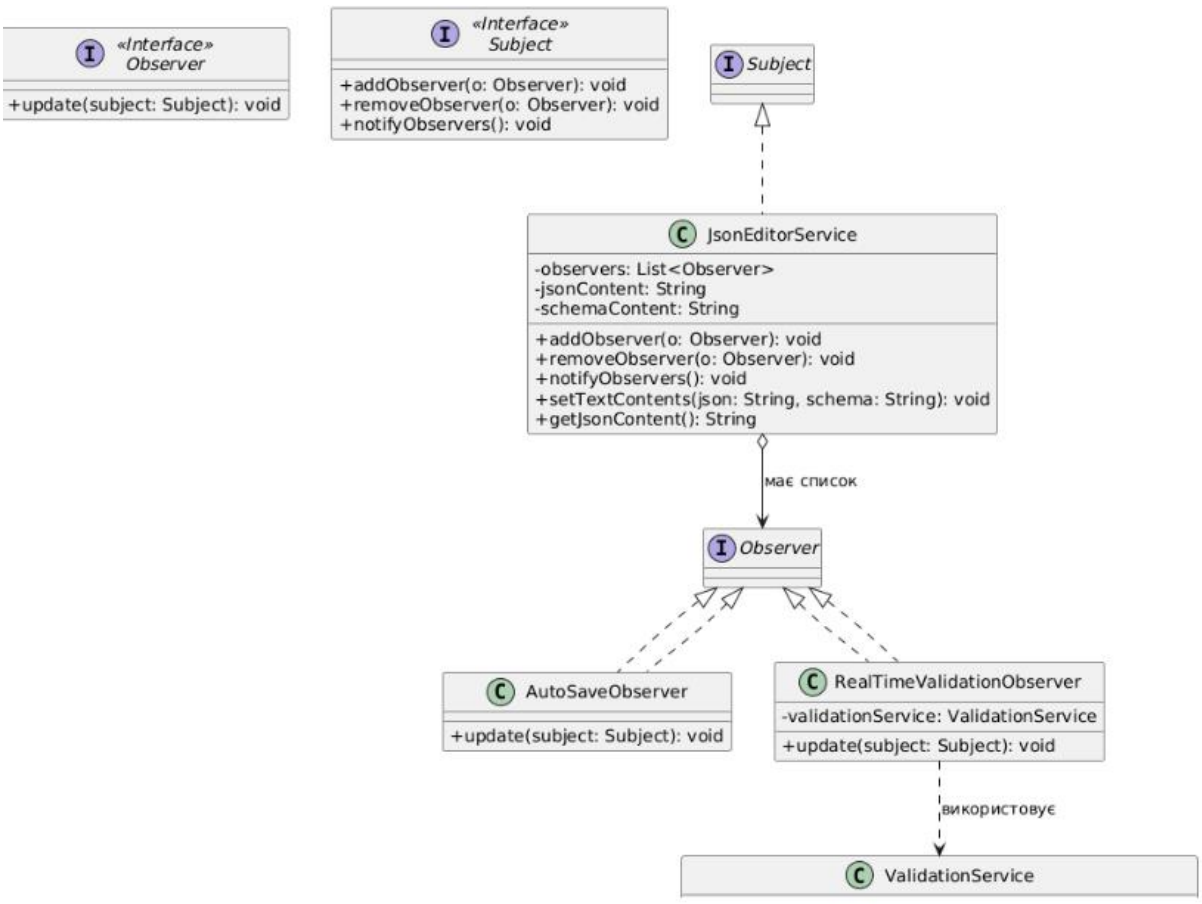
Створимо конкртених спостерігачів: AutoSaveObserver та RealTimeValidationObserver

```java
package com.example.jsontool.observer;

import com.example.jsontool.service.JsonEditorService;
import org.springframework.stereotype.Component;

@Component  new *
public class AutoSaveObserver implements Observer {

    // @Autowired
    // private HistoryService historyService;

    @Override  1 usage  new *
    public void update(Subject subject) {
        if (subject instanceof JsonEditorService) {
            JsonEditorService editor = (JsonEditorService) subject;
            String contentToSave = editor.getJsonContent();

            // historyService.saveHistory(contentToSave);

            System.out.println("[AutoSaveObserver]: Зміст оновлено! Виконую автозбереження...");
        }
    }
}
```

AutoSaveObserver.java
Реагує на зміни в редакторі та виконує автозбереження.

```java
@Component  new *
public class RealTimeValidationObserver implements Observer {

    @Autowired
    private ValidationService validationService;

    @Override  new *
    public void update(Subject subject) {
        if (subject instanceof JsonEditorService) {
            JsonEditorService editor = (JsonEditorService) subject;
            String json = editor.getJsonContent();
            String schema = editor.getSchemaContent();

            Validation strategy = (schema == null || schema.trim().isEmpty())
                    ? new SyntaxValidation()
                    : new SchemaValidation();

            List<String> errors = validationService.executeValidation(strategy, json, schema);

            if (errors.isEmpty()) {
                System.out.println("[ValidationObserver]: Текст оновлено. Валідація успішна.");
            } else {
                System.out.println("[ValidationObserver]: Текст оновлено. Знайдено помилки: " + errors);
            }
        }
    }
}
```

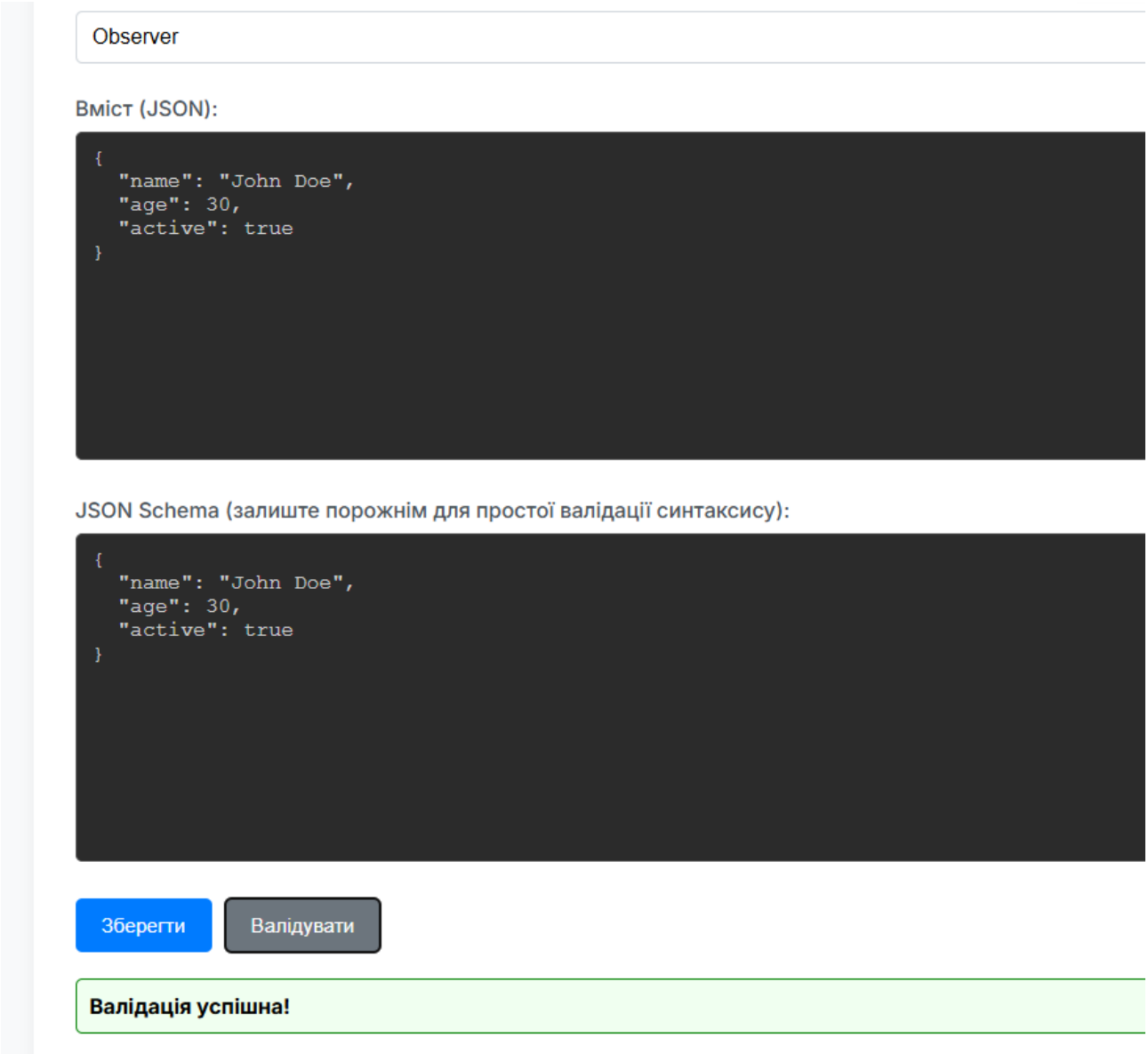
RealTimeValidationObserver.java
Реагує на зміни в редакторі та запускає валідацію.



Діаграма класів, для реалізування патерну «Observer»

Демонастрація роботи патерну

```
2025-11-15T01:23:09.478+02:00  INFO 15064 --- [json-tool] [
--- [Observer Pattern] Реєстрація спостерігачів... ---
Зареєстровано: AutoSaveObserver
Зареєстровано: RealTimeValidationObserver
--- Реєстрацію завершено ---
2025-11-15T01:23:09.842+02:00  WARN 15064 --- [json-tool] [
2025-11-15T01:23:10.352+02:00  INFO 15064 --- [json-tool] [
2025-11-15T01:23:10.358+02:00  INFO 15064 --- [json-tool] [
```




```
2025-11-15T01:24:54.540+02:00  WARN 15064 --- [json-tool] [nio
[ValidationObserver]: Текст оновлено. Валідація успішна.
```

5)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 шаблон

Висновки

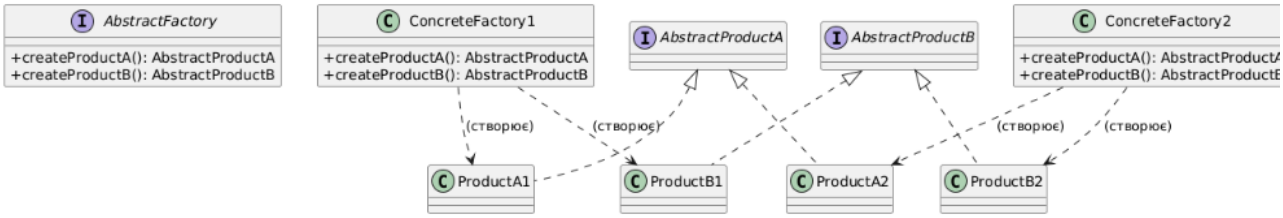
Під час виконання лабораторної роботи №6 я вивчив патерн **«Observer»**. Метою було реалізувати цей патерн для додавання динамічної функціональності, зокрема, для реалізації вимоги "Auto save... history" з теми мого проєкту. Для цього було створено інтерфейси Subject та Observer, а також конкретний "Суб'єкт" — JsonEditorService. Було реалізовано два "Спостерігачі": AutoSaveObserver та RealTimeValidationObserver, які "підписуються" на JsonEditorService при старті програми. Тепер, коли DocumentController оновлює стан у JsonEditorService, він автоматично сповіщає всіх спостерігачів, які виконують свою логіку (наприклад, логують автозбереження), не будучи напряму пов'язаними з контролером.

Відповіді на контрольні питання

1. Яке призначення шаблону «Абстрактна фабрика»?

«Абстрактна фабрика» (Abstract Factory) — це породжуючий патерн, який надає інтерфейс для створення **сімейств взаємопов'язаних об'єктів**, не вказуючи їхні конкретні класи. Це дозволяє створювати набори об'єктів, які гарантовано сумісні один з одним (наприклад, кнопки та чекбокси в стилі Windows або в стилі macOS).

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

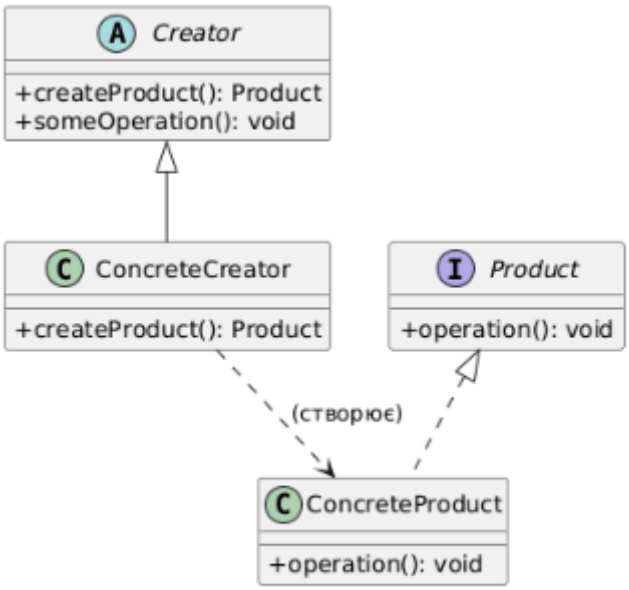
1. **AbstractFactory (Абстрактна фабрика):** Інтерфейс, що оголошує методи для створення кожного продукту з сімейства (напр., createButton(), createCheckbox()).
2. **ConcreteFactory (Конкретна фабрика):** Класи, що реалізують AbstractFactory. Кожна фабрика створює продукти певної варіації (напр., WinFactory створює WinButton і WinCheckbox).
3. **AbstractProduct (Абстрактний продукт):** Інтерфейси для кожного типу продукту (напр., Button, Checkbox).
4. **ConcreteProduct (Конкретний продукт):** Класи, що реалізують інтерфейси AbstractProduct (напр., WinButton, MacCheckbox).

Взаємодія: Клієнт отримує екземпляр ConcreteFactory і використовує її методи для створення продуктів (AbstractProduct). Клієнт працює з продуктами через їхні абстрактні інтерфейси, не знаючи, яку конкретну реалізацію він отримав.

4. Яке призначення шаблону «Фабричний метод»?

«Фабричний метод» (Factory Method) — це породжуючий патерн, який визначає **інтерфейс для створення об'єкта**, але дозволяє **підкласам вирішувати, який саме клас створювати**. Він делегує створення об'єктів своїм підкласам.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

1. **Product (Продукт):** Інтерфейс для об'єктів, які створюються.
2. **ConcreteProduct (Конкретний продукт):** Класи, що реалізують інтерфейс Product.
3. **Creator (Творець):** Абстрактний клас (або інтерфейс), що оголошує factoryMethod(), який повертає Product.
4. **ConcreteCreator (Конкретний творець):** Клас, що успадковує Creator і перевизначає factoryMethod(), щоб створювати ConcreteProduct.

Взаємодія: Клієнт викликає factoryMethod() у ConcreteCreator, щоб отримати Product.

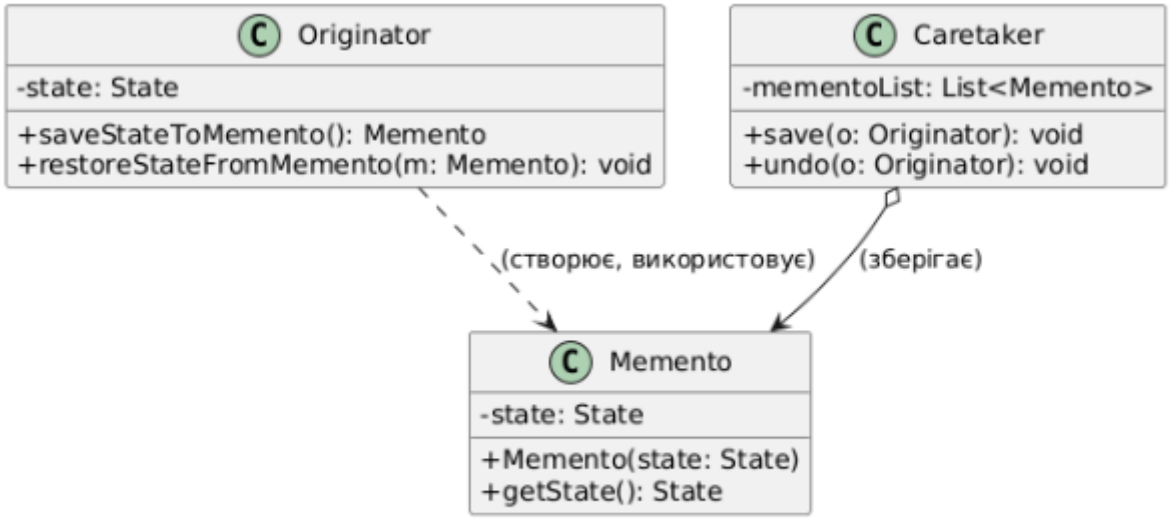
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

- **Мета:** "Фабричний метод" — це **один метод** для створення **одного об'єкта**. "Абстрактна фабрика" — це **об'єкт** з **багатьма методами** для створення **сімейства пов'язаних об'єктів**.
- **Реалізація:** "Фабричний метод" зазвичай реалізується через **успадкування** (підкласи перевизначають метод). "Абстрактна фабрика" частіше реалізується через **композицію** (клієнт отримує об'єкт-фабрику і працює з ним).

8. Яке призначення шаблону «Знімок»?

Шаблон «Знімок» (Memento) — це поведінковий патерн, який дозволяє **зберігати та відновлювати попередній стан об'єкта**, не порушуючи його інкапсуляцію.

9. Нарисуйте структуру шаблону «Знімок».



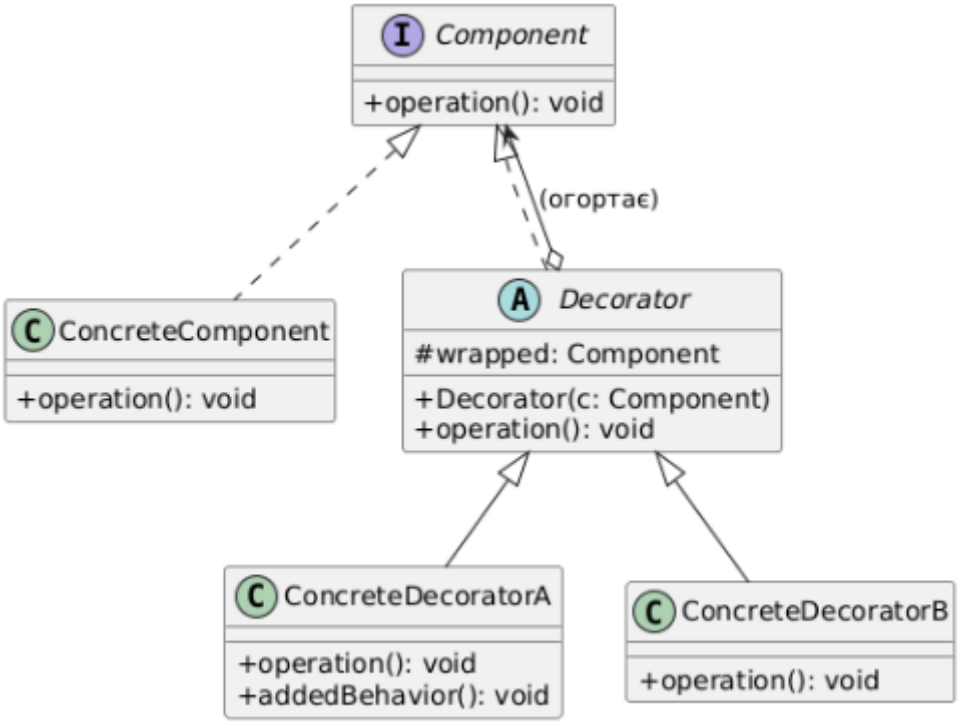
10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

1. **Originator (Творець):** Об'єкт, стан якого потрібно зберегти. Він вміє створювати Memento зі своїм станом і відновлювати стан з Memento.
2. **Memento (Знімок):** Простий об'єкт, що зберігає стан Originator. Він має бути захищеним (часто робиться внутрішнім класом Originator).
3. **Caretaker (Опікун):** Клас, що відповідає за зберігання "стопки" знімків. Він не знає, що всередині Memento, він просто зберігає їх.

11. Яке призначення шаблону «Декоратор»?

«Декоратор» (Decorator) — це структурний патерн, який дозволяє **динамічно додавати нову поведінку (функціональність) об'єктам**, "огортаючи" їх у класи-декоратори. Це гнучка альтернатива успадкуванню.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

1. **Component (Компонент):** Загальний інтерфейс для об'єктів, які ми "огортаємо".
2. **ConcreteComponent (Конкретний компонент):** Базовий клас, якому ми додаємо функціональність.
3. **Decorator (Декоратор):** Абстрактний клас, що реалізує Component. Він містить посилання на об'єкт Component, який він "огортає".
4. **ConcreteDecorator (Конкретний декоратор):** Класи, що додають конкретну нову поведінку.

Взаємодія: Клієнт працює з об'єктом через інтерфейс Component. Коли клієнт викликає operation(), ConcreteDecorator спочатку робить свою додаткову роботу, а потім **делегує** виклик "огорнутому" об'єкту.

14. Які є обмеження використання шаблону «декоратор»?

- **Багато дрібних об'єктів:** Архітектура може "засмітитися" великою кількістю маленьких класів-декораторів.
- **Складність конфігурації:** Іноді складно зібрати об'єкт з потрібними декораторами у правильному порядку.
- **Не можна додати новий метод:** Декоратор може лише *перевизначити* поведінку існуючих методів інтерфейсу, але не може додати нові публічні методи.