Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №6

Технологія розробки програмного забезпечення

«ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator».

Мета: ознайомитись з шаблонами «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувати один з шаблонів при реалізації програми.

Хід роботи:

Варіант:

..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.

Згідно з моїм варіантом, необхідно реалізувати шаблон Observer. Шаблон "Observer" визначає залежність одного або декількох об'єктів від змін у стані іншого об'єкта та автоматично повідомляє їх про будь-які зміни. Основні учасники цього шаблону - Subject (спостережуваний об'єкт) і Observer (спостерігач).

В моєму проекті я реалізувала цей шаблон для виводу результатів валідації після натиснення на кнопку «Валідувати». Реалізація цього патерну знаходиться в директорії observer_pattern.

observer це інтерфейс, який декларує метод update. Кожен конкретний клас спостерігач повинен реалізувати цей інтерфейс.

```python
from abc import ABC, abstractmethod


2 usages
class Observer(ABC):
    1 usage (1 dynamic)
    @abstractmethod
    def update(self, message):
        pass
```

validate_button_observer є конкретною реалізацією спостерігача. Коли змінюється стан спостережуваного об’єкта (ValidateButtonSubject) він викликає метод create_validation_result_view_window, у якому описана логіка створення вікна для виводу результатів валідації.

validate_button_observer.py × validate_json_command.py json_schema_service.py I_history_se

```python
from app.views.observer_pattern.observer import Observer
from tkinter import *


2 usages
class ValidateButtonObserver(Observer):
    1 usage (1 dynamic)
    def update(self, message):
        # Викликаємо метод для створення нового вікна
        self.create_validation_result_view_window(message)

    1 usage
    def create_validation_result_view_window(self, message):
        # Створюємо нове вікно
        root = Tk()
        # Додаємо заголовок вікна
        root.title("Вікно з виводом тексту")
        # Створюємо віджет Text для виводу тексту
        text_widget = Text(root, height=10, width=40)
        # Вставляємо текст (результати валідації)
        text_widget.insert(END, message)
        # Встановлюємо режим DISABLED, щоб текст не можна було редагувати
        text_widget.config(state=DISABLED)
        text_widget.pack()
```

Subject це клас який описує логіку роботи з спостерігачами. Він містить масив спостерігачів, а також методи для додавання видалення та сповіщення спостерігачів.

```
3 usages
class Subject:
    def __init__(self):
        self._observers = []

    1 usage
    def add_observer(self, observer):
        if observer not in self._observers:
            self._observers.append(observer)

    def remove_observer(self, observer):
        self._observers.remove(observer)

    1 usage
    def notify_observers(self, message):
        for observer in self._observers:
            observer.update(message)
```

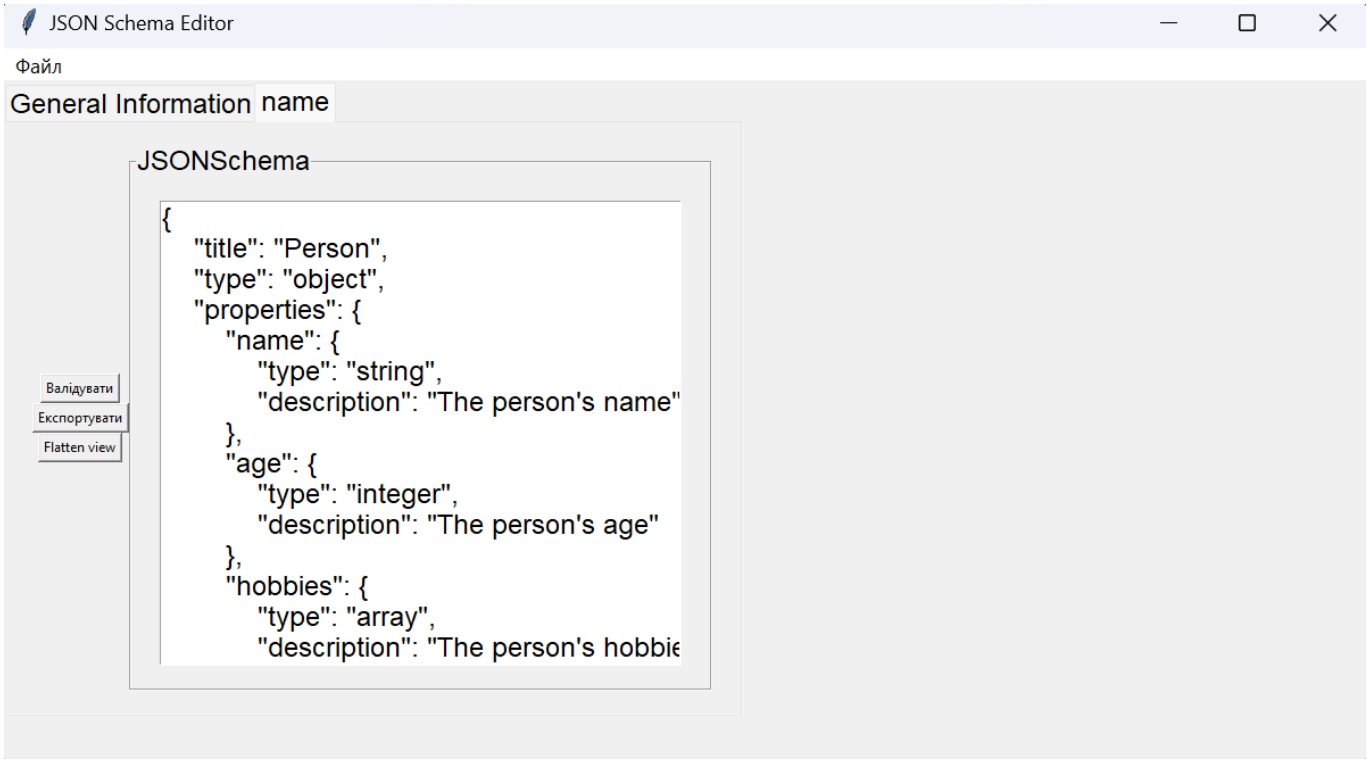
Клас validate_button_subject це клас, який успадковується від tk.Button та є спостережуваним об’єктом. Цей клас отримує в параметри вікно в якому має бути розміщена кнопка, підпис кнопки, а також команду яку має вона виконувати. Далі створюється кнопка з цими параметрами та під час її натискання викликається метод press_button. В цьому методі викликається метод execute для отриманої в параметрі команди та результат виконання записується в змінну, яка передається в метод notify_observers. В цій змінній збережені результати валідації json.

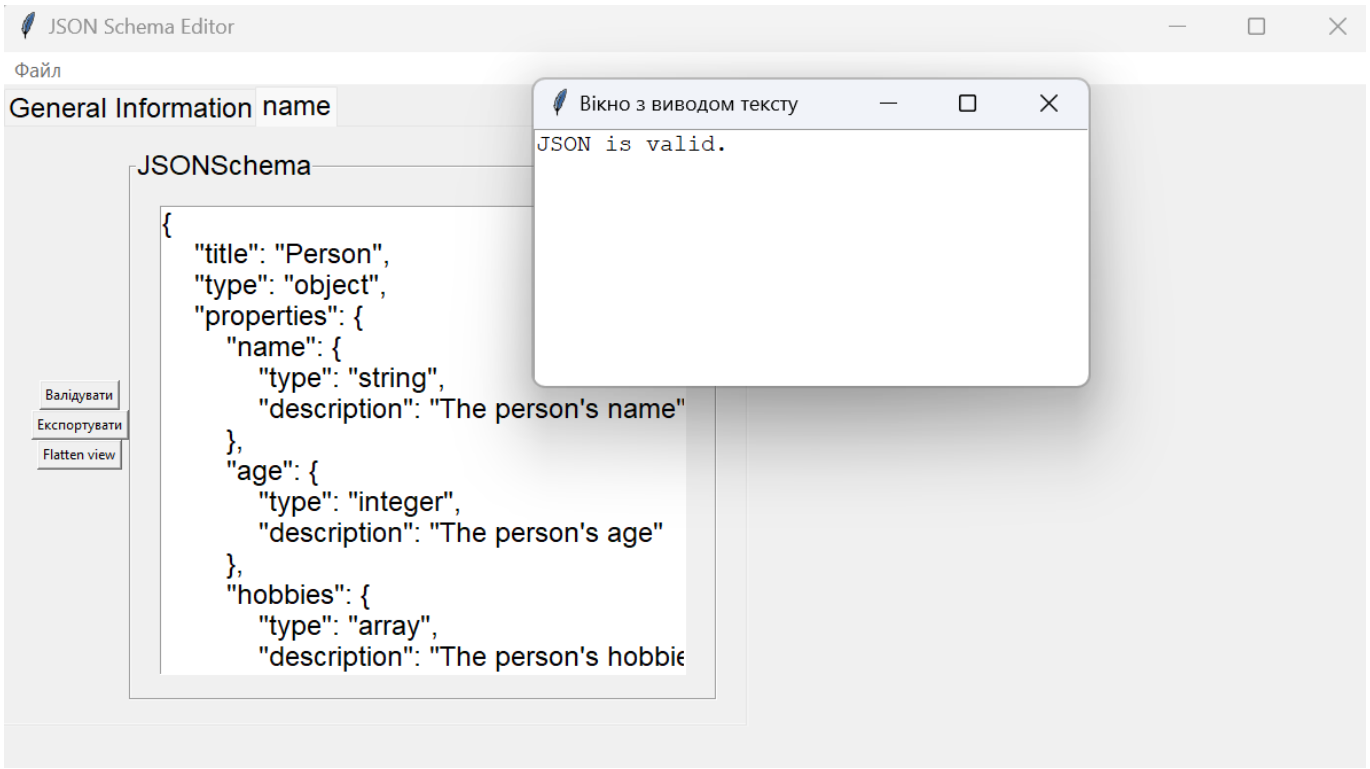
```python
from tkinter import *
from app.views.command_pattern.interface_command import InterfaceCommand
from app.views.observer_pattern.subject import Subject


2 usages
class ValidateButtonSubject(Button, Subject):
    def __init__(self, main_frame, text, _command: InterfaceCommand):
        Subject.__init__(self)
        Button.__init__(self)
        self.button = Button(main_frame, text=text, command=self.press_button)
        self.button.pack()
        self.main_frame = main_frame
        self._command = _command

    1 usage
    def press_button(self):
        print("Кнопка валідації була натиснута.")
        message = self._command.execute()
        # Викликаємо метод notify_observers
        self.notify_observers(message)
```

Створення спостерігача та спостережуваного об’єкта знаходиться в класі toolbar_view.

```python
# Створюємо команду валідації
validate_json_command = ValidateJSONCommand(schema_text)
# Додаємо кнопку для валідації JSON-схеми
self.validate_button = ValidateButtonSubject(self.toolbar, text: "Валідувати", validate_json_command)
# Створюємо спостерігача
observer = ValidateButtonObserver()
# Додаємо спостерігача для
self.validate_button.add_observer(observer)
```

В результаті виконання коду маємо такий результат:





Висновок: на даній лабораторній роботі я ознайомилась з шаблонами «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator», реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувала один з шаблонів при реалізації програми.