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 №7

Технологія розробки програмного забезпечення

«ШАБЛОНИ «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»»

Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони «Mediator», «Facade», «Bridge», «Template method».

Мета: ознайомитись з шаблонами «Mediator», «Facade», «Bridge», «Template method», 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:

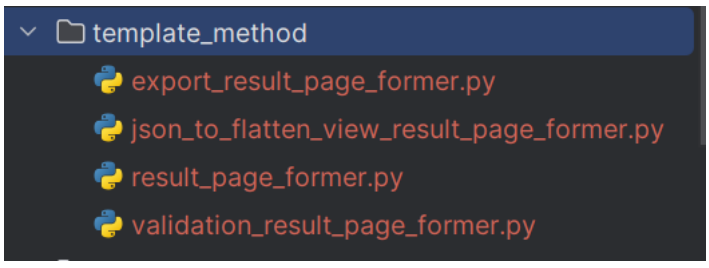
..28 JSON Tool (ENG) (strategy, command, observer, template method, flyweight)

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 Template method.

Його основна ідея полягає в тому, щоб визначити скелет алгоритму в базовому класі, а деякі кроки цього алгоритму дозволити покладати на підкласи. Таким чином, можна перевизначити конкретні кроки алгоритму в підкласах без зміни його загальної структури.

Даний патерн реалізований в дерикторії template_method:



ResultPageFormer це абстрактний клас, який визначає скелет (структуру) алгоритму для створення сторінки результату натиснення певних кнопок. Він містить метод execute(), який описує порядок виклику методів. Також він містить

метод open_result_page(), який містить реалізацію за замовчуванням. Метод create_page_content() є абстактним і повинен перевизначатись в кожному класі наслідникy.

```python
from abc import ABC, abstractmethod
from tkinter import *
import ctypes

class ResultPageFormer(ABC):
    def execute(self, text):
        root = self.open_result_page()
        self.create_page_content(root, text)

    def open_result_page(self):
        # Створюємо нове вікно
        root = Tk()
        # Додаємо заголовок вікна
        root.title("JSON Editor")
        # Встановлюємо розмір вікна
        root.geometry("750x500")
        # Встановлюємо DPI-свідомість процесу в Windows, для того щоб програма могла змінювати
        # масштаб елементів інтерфейсу користувача, щоб вони виглядали на екрані більш чітко
        ctypes.windll.shcore.SetProcessDpiAwareness(True)
        return root

    @abstractmethod
    def create_page_content(self, root, text):
        pass
```

Клас ValidationResultPageFormer є реалізацією абстрактного класу ResultPageFormer. Він реалізує метод create_page_content() у якому ми створюємо вікно для виводу результатів валідації.

```python
from app.views.template_method.result_page_former import ResultPageFormer
from tkinter import *


class ValidationResultPageFormer(ResultPageFormer):
    def create_page_content(self, root, message):
        # Створюємо віджет Text для виводу тексту
        text_widget = Text(root, height=20, width=60)
        # 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()
```

Клас JSONToFlattenViewResultPageFormer також реалізує абстрактний клас ResultPageFormer. Він реалізує метод create_page_content() у якому ми створюємо вікно для виводу результатів перетворення json.

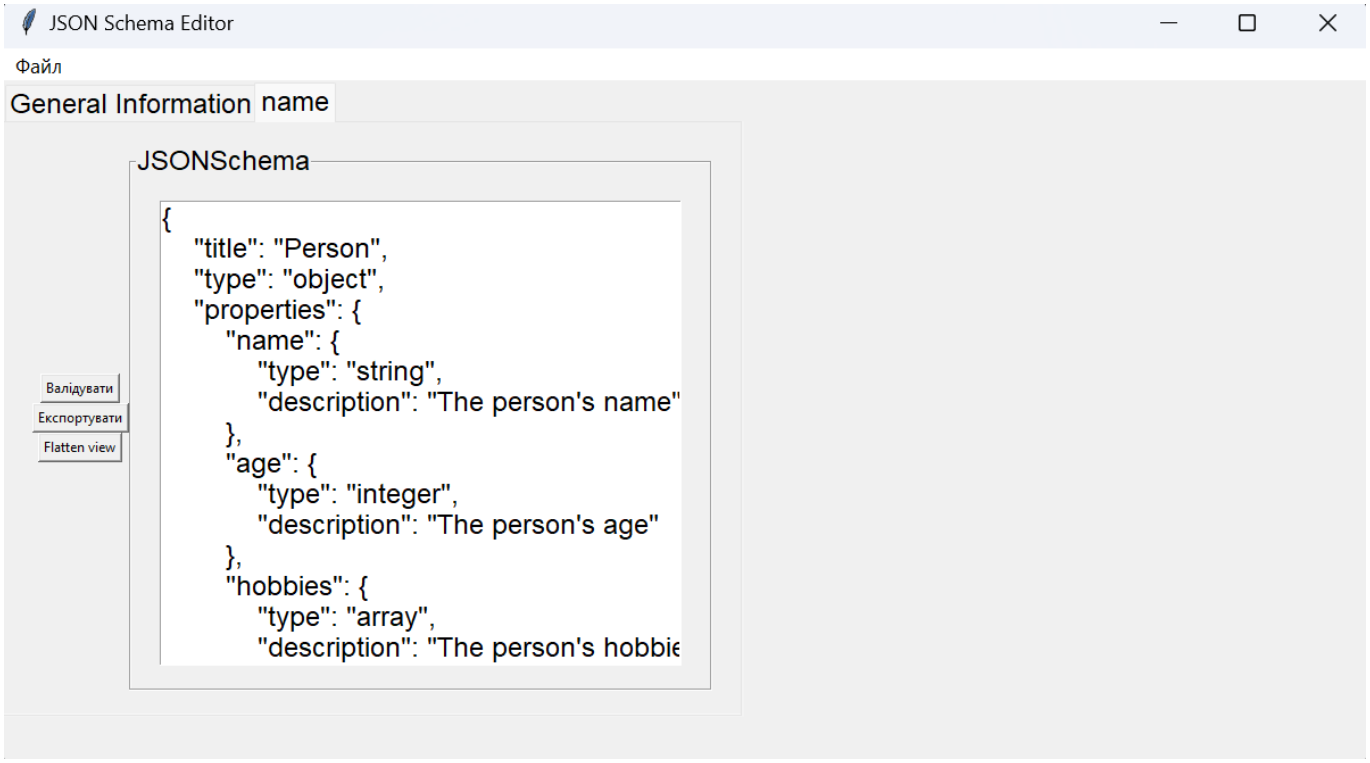
```python
from app.views.template_method.result_page_former import ResultPageFormer
from tkinter import *


2 usages
class JSONToFlattenViewResultPageFormer(ResultPageFormer):
    def create_page_content(self, root, json):
        main_frame = LabelFrame(root, text="JSON to flatten view")
        # Створюємо віджет Text для виводу тексту
        text_widget = Text(main_frame, height=20, width=80)
        # Вставляємо текст
        text_widget.insert(END, json)
        # Встановлюємо режим DISABLED, щоб текст не можна було редагувати
        text_widget.config(state=DISABLED)
        text_widget.pack()
        main_frame.pack()
```

Створення об’єктів відбувається в спостерігачах та має наступний вигляд:

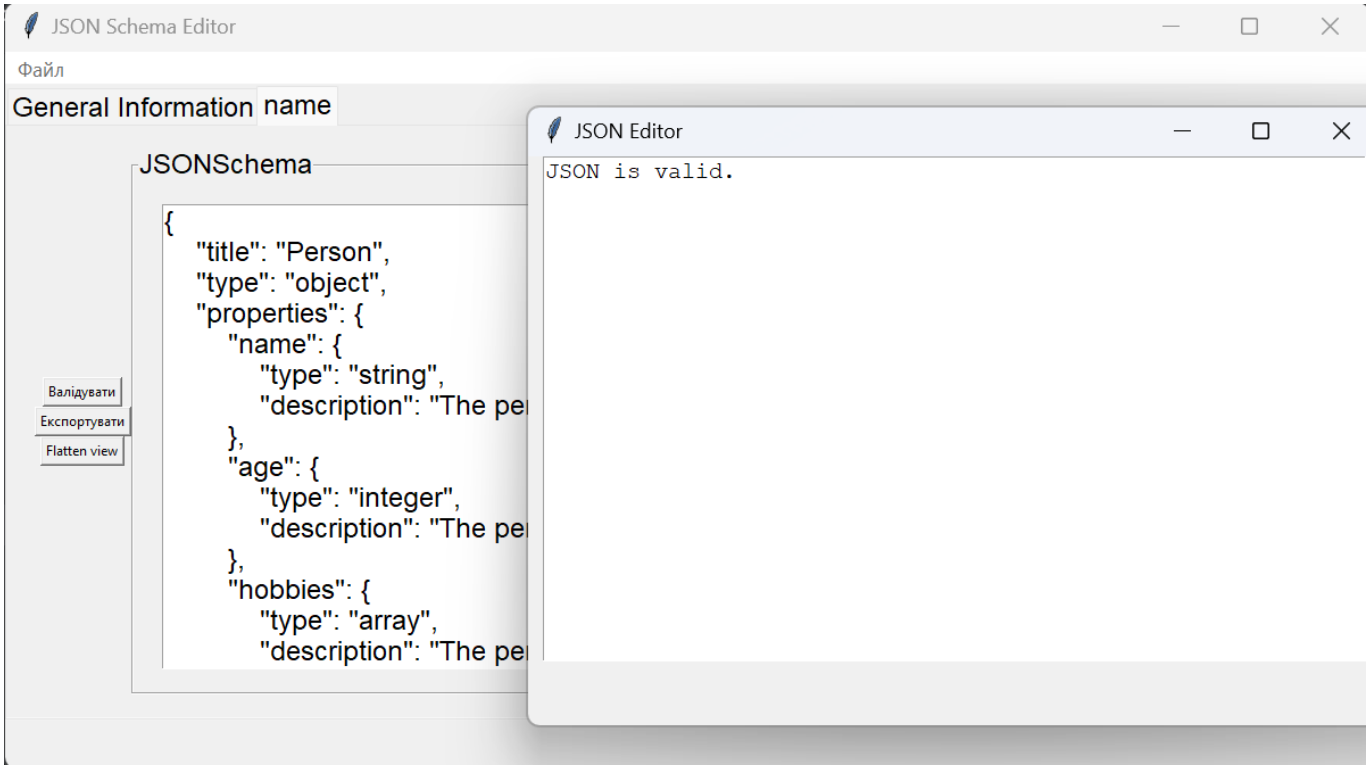
```python
from app.views.observer_pattern.observer import Observer
from app.views.template_method.export_result_page_former import ExportResultPageFormer


2 usages
class ExportButtonObserver(Observer):
    1 usage (1 dynamic)
    def update(self, message):
        # Створюємо об'єкт класу ValidationResultPageFormer
        result_page_former = ExportResultPageFormer()
        # Викликаємо метод execute()
        result_page_former.execute(message)
```

Тобто алгоритм дій такий: коли ми натискаємо на якусь кнопку, то її спостерігач отримує повідомлення про те що кнопка натиснута і створює об’єкт для відображення результатів (наприклад, результатів валідації).

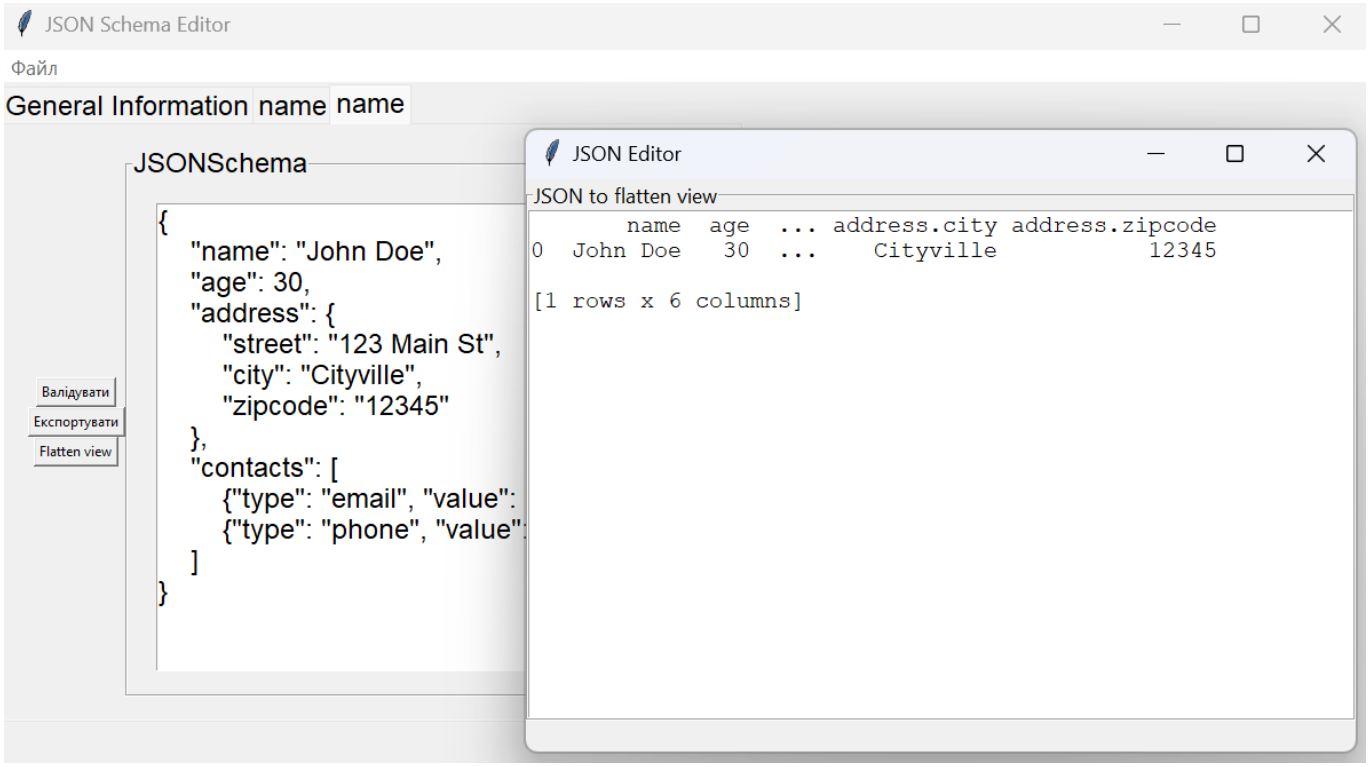
В результаті виконання коду маємо такий результат:



Натискаємо на кнопку валідації:



Натискаємо на кнопку Flatten view:



Висновок: на даній лабораторній роботі я ознайомилась з шаблонами «Mediator», «Facade», «Bridge», «Template method», реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувала один з шаблонів при реалізації програми.