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 №4

Технологія розробки програмного забезпечення

«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY».»

Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони “Sindleton”, “Iterator”, “Proxy”, “State”, “Strategy”

Мета: ознайомитись з шаблонами “Sindleton”, “Iterator”, “Proxy”, “State”, “Strategy”, 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:

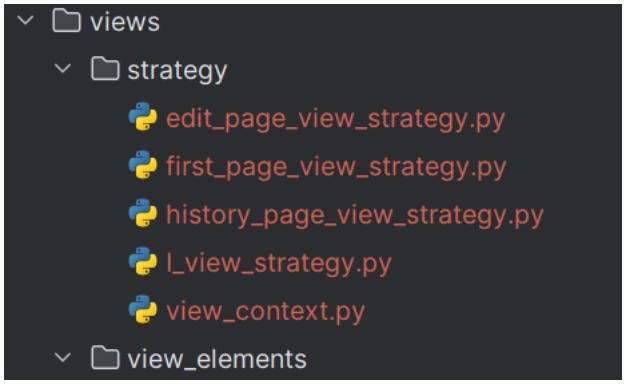
Варіант:

..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 Strategy. Суть цього шаблону полягає в тому що ми можемо створювати декілька алгоритмів поведінки об’єкта. Тобто кожен алгоритм має власну реалізацію визначену в окремому класі та можуть бути взаємозамінні в об’єкті що їх використовує. Даний шаблон застосовується коли існують різні способи обробки інформації.

В моєму проекті я реалізувала цей шаблон для відображення сторінок програми.



У нас є інтерфейс interface_view_strategy який описує методи, які повинні реалізовувати класи, які його імплементують. В нашому випадку це метод create_operating_window, який повинен створювати вікно для роботи.

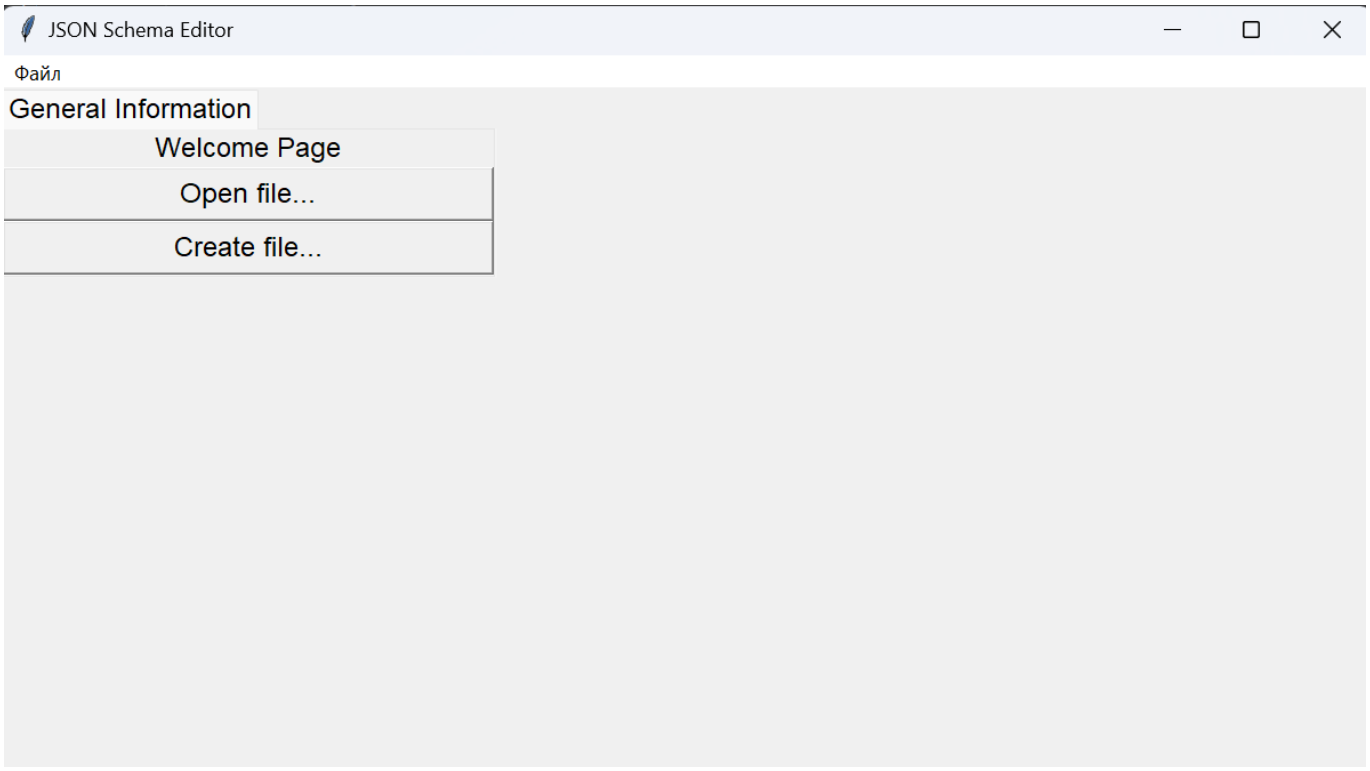
```python
from abc import ABC, abstractmethod


9 usages  Valeriia Khyzhniak *
class InterfaceViewStrategy(ABC):
    1 usage  Valeriia Khyzhniak
    @abstractmethod
    def create_operating_window(self, window):
        pass
```

Класи FirstPageViewStrategy імплементує інтерфейс interface_view_strategy та реалізовує метод create_operating_window, який створює початкову сторінку.

```python
from tkinter import *
from app.views.create_new_page import CreateNewPage
from app.views.strategy.interface_view_strategy import InterfaceViewStrategy


2 usages  Valeriia Khyzhniak *
class FirstPageViewStrategy(InterfaceViewStrategy):

    # Створюємо об'єкт класу CreateNewPage()
    create_new_window = CreateNewPage()
    font = ('Consolas 20', 18)
    width = 30

    # Ініціалізуємо атрибути класу
    Valeriia Khyzhniak
    def __init__(self):
        super().__init__()
```

```python
def create_operating_window(self, window):
    # Створіть власний стиль для вкладок
    first_page_frame = Frame(window, width=100, height=700)
    first_page_label = Label(first_page_frame, text='Welcome Page', font=self.font)
    # Створюємо кнопки
    open_file_button = Button(first_page_frame, width=self.width, text="Open file...", font=self.font
                              , command=lambda: self.create_new_window.open_new_window(window)
                              )
    create_file_button = Button(first_page_frame, width=self.width, text="Create file...", font=self.font
                                # , command=lambda: self.create_new_window.create_new_window()
                                )
    # Розташовуємо їх їх
    first_page_label.grid(row=0, column=0)
    open_file_button.grid(row=1, column=0)
    create_file_button.grid(row=2, column=0)
    first_page_frame.pack(fill='both', expand=True)
    # Додаємо вкладку
    window.add(first_page_frame, text='General Information')
    # Розташовуємо вкладку вгорі сторінки
    window.place(relx=0, rely=0, anchor='nw')
    return window
```

Результат роботи цього класу має наступний вигляд:



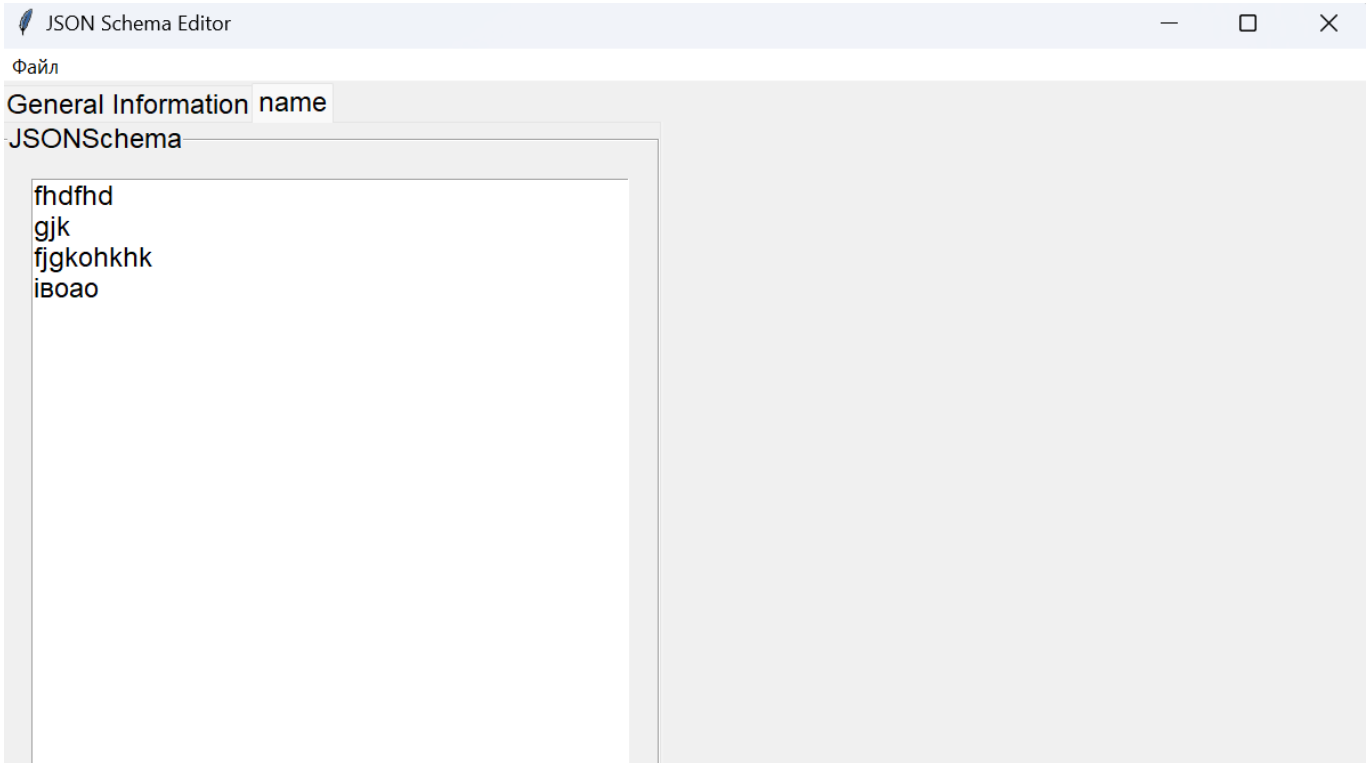
Клас EditPageViewStrategy імплементує інтерфейс interface_view_strategy та реалізовує метод create_operating_window, який створює сторінку для редагування json.

```python
import tkinter as tk
from tkinter import *
from app.views.strategy.interface_view_strategy import InterfaceViewStrategy


7 usages  Valeriia Khyzhniak *
class EditPageViewStrategy(InterfaceViewStrategy):
    font = ('Consolas 20', 18)
    # Створюємо змінну для збереження шляху відкритого файлу
    _opened_file = ''
    # Створюємо змінну для збереження вікна з текстом
    _schema_text = Text

    # Ініціалізуємо атрибути класу
    Valeriia Khyzhniak
    def __init__(self):
        super().__init__()
```

```python
    def create_operating_window(self, edit_page):
        # Створюємо об'єкт рамки
        self.schema_frame = LabelFrame(edit_page, width=100, height=70, text='JSONSchema', padx=25, pady=20,
                                       font=self.font)
        # Створюємо об'єкт текстового поля для редагування JSON-схеми
        self.schema_text = Text(self.schema_frame, font=self.font, width=40, height=20, wrap=tk.NONE)
        # Розташовуємо текстове поле у рамці
        self.schema_text.pack(fill=BOTH, expand=True)
        # Розташовуємо рамку у лівій частині вікна
        self.schema_frame.pack(side=LEFT, fill=BOTH, expand=True)
        edit_page.add(self.schema_frame, text='name')
        # Розташуємо вкладку вгорі сторінки
        edit_page.place(relx=0, rely=0, anchor='nw')
        return self.schema_text
```

```python
    @property
    def opened_file(self):
        return self._opened_file

    1 usage  Valeriia Khyzhniak
    @opened_file.setter
    def opened_file(self, value):
        self._opened_file = value

    4 usages  Valeriia Khyzhniak
    @property
    def schema_text(self):
        return self._schema_text

    2 usages  Valeriia Khyzhniak
    @schema_text.setter
    def schema_text(self, value):
        self._schema_text = value
```

Результати роботи цього класу має наступний вигляд:



Клас ViewContext отримує в конструктор параметр стратегію та викликає метод створення сторінки для цієї стратегії.

```python
from app.views.strategy.interface_view_strategy import InterfaceViewStrategy


4 usages  Valeriia Khyzhniak *
class ViewContext:
    strategy = InterfaceViewStrategy

    Valeriia Khyzhniak *
    def __init__(self, strategy: InterfaceViewStrategy):
        super().__init__()
        self.strategy = strategy

    2 usages  Valeriia Khyzhniak
    def view_page_strategy(self, page):
        return self.strategy.create_operating_window(page)
```

Стратегія FirstPageViewStrategy викликається під час створення редактора в класі JSONSchemaView

```python
    # Створюємо початкову сторінку
    self.welcome_page = ViewContext(FirstPageViewStrategy()).view_page_strategy(self.window)
```

Стратегія EditPageViewStrategy викликається в класі CreateNewPage після натискання на кнопку Open_file.

```python
from app.services.json_schema_service import JSONSchemaService
from app.views.strategy.edit_page_view_strategy import EditPageViewStrategy
from app.views.strategy.view_context import ViewContext


4 usages  Valeriia Khyzhniak
class CreateNewPage:
    json_schema_service = JSONSchemaService()

    Valeriia Khyzhniak
    def __init__(self):
        super().__init__()

    2 usages  Valeriia Khyzhniak
    def open_new_window(self, window):
        # Створюмо сторінку для редагування
        schema_text = ViewContext(EditPageViewStrategy()).view_page_strategy(window)
        # Відкриваємо в неї вміст файлу
        op_file = self.json_schema_service.open_file(schema_text)
        # Записуємо значення в сеттери
        EditPageViewStrategy.opened_file = op_file
        EditPageViewStrategy.schema_text = schema_text
```

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

```python
from I_history_repository import IHistoryRepository
from app.models.json_history import JSONHistory


Valeriia Khyzhniak *
class HistoryRepository(IHistoryRepository):
    Valeriia Khyzhniak *
    def __init__(self, connection):
        super().__init__()
        self.connection = connection
        # Створюємо курсор для БД
        self.cursor = connection.cursor()
        # Створюємо таблицю json_schemas якщо її не існує
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS json_histories (
                        history_id INTEGER PRIMARY KEY,
                        json_id INTEGER,
                        date_of_saving DATE,
                    )""")
        connection.commit()
```

```
    Valeriia Khyzhniak *
19  def delete_all(self):
20      with self.connection.cursor() as cursor:
21          cursor.execute('DELETE FROM json_histories')
22      self.connection.commit()
23
```

```
24  def get_by_id(self, history_id):
25      # Знаходимо елемент із заданою id
26      self.cursor.execute("SELECT * FROM json_histories WHERE history_id=?", (history_id,))
27      # Записуємо його в результат
28      result = self.cursor.fetchone()
29      if result:
30          # Повертаємо знайдений елемент
31          return JSONHistory(result[0], result[1], result[2])
32      else:
33          return None
34
    new *
35  def _insert(self, json_history):
36      self.cursor.execute(
37          "INSERT INTO json_histories (history_id, json_id, date_of_saving) VALUES (?, ?, ?)"
38          , (json_history.history_id, json_history.json_id, json_history.date_of_saving))
39      self.connection.commit()
```

```
    def delete_by_id(self, history_id):
        self.cursor.execute("DELETE FROM json_histories WHERE history_id=?", (history_id,))
        self.connection.commit()

    new *
    def _update(self, json_history):
        self.cursor.execute("UPDATE json_histories SET json_id=?, date_of_saving=? WHERE history_id=?",
                            (json_history.json_id, json_history.date_of_saving, json_history.history_id))
        self.connection.commit()
```

```python
from app.repositories.I_json_schema_repository import IJSONSchemaRepository
from app.models.json_schema import JSONSchema


Valeriia Khyzhniak *
class JSONSchemaRepository(IJSONSchemaRepository):
    Valeriia Khyzhniak *
    def __init__(self, connection):
        super().__init__()
        self.connection = connection
        # Створюємо курсор для БД
        self.cursor = connection.cursor()
        # Створюємо таблицю json_schemas якщо її не існує
        self.cursor.execute("""CREATE TABLE IF NOT EXISTS json_schemas (
            json_id INTEGER PRIMARY KEY,
            json_name text,
            json_data text,
            json_file_path text
        )""")
        connection.commit()
```

```python
    def get_by_id(self, json_id):
        # Знаходимо елемент із заданою id
        self.cursor.execute("SELECT * FROM json_schemas WHERE json_id=?", (json_id,))
        # Записуємо його в результат
        result = self.cursor.fetchone()
        if result:
            # Повертаємо знайдений елемент
            return JSONSchema(result[0], result[1], result[2], result[3])
        else:
            return None

    new *
    def _insert(self, json_schema):
        self.cursor.execute("INSERT INTO json_schemas (json_id, json_name, json_data, json_file_path) VALUES (?, ?, ?, ?)"
                            , (json_schema.json_id, json_schema.json_name, json_schema.json_data,
                               json_schema.json_file_path,))
        self.connection.commit()
```

```
    def _update(self, json_schema):
        self.cursor.execute("UPDATE json_schemas SET json_name=?, json_data=?, json_file_path=? WHERE json_id=?",
                            (json_schema.json_name, json_schema.json_data, json_schema.json_file_path,
                             json_schema.json_id))
        self.connection.commit()

    new *
    def delete_by_id(self, json_id):
        self.cursor.execute("DELETE FROM json_schemas WHERE json_id=?", (json_id,))
        self.connection.commit()

    new *
    def get_by_name(self, json_name):
        self.cursor.execute("SELECT * FROM json_schemas WHERE json_name=?", (json_name,))
        result = self.cursor.fetchone()
        if result:
            return JSONSchema(result[0], result[1], result[2], result[3])
        else:
            return None
```

Висновок: на даній лабораторній роботі я ознайомилась з шаблонами “Sindleton”, “Iterator”, “Proxy”, “State”, “Strategy”, реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувала один з шаблонів при реалізації програми.