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:

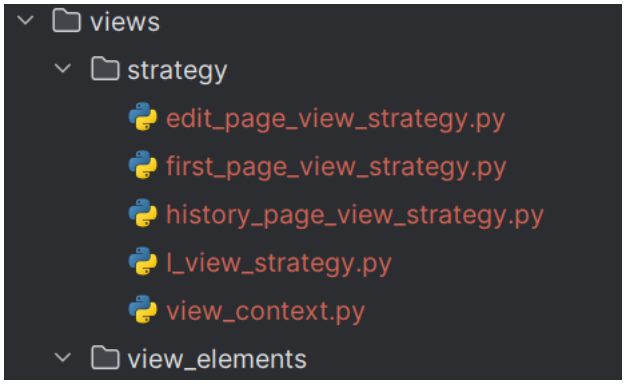
Варіант:

..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 I_view_strategy який описує методи які повинні реалізовувати класи, які його імплементують. Класи HistoryPageViewStrategy, FirstPageViewStrategy та EditPageViewStrategy імплементуть цей інтерфейс та описують логіку створення сторінки історії, першої сторінки та сторінки для редагування відповідно. Клас ViewContext отримує в конструктор параметр стратегія та викликає метод створення сторінки для певної стратегії.

database.py edit_page_view_strategy.py

```python
import tkinter as tk
from tkinter import *
import ctypes
from tkinter import ttk

from app.views.strategy.I_view_strategy import IViewStrategy


9 usages
class EditPageViewStrategy(IViewStrategy):
    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
    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
```

```
        2 usages
35      @property
36      def opened_file(self):
37          return self._opened_file
38
        1 usage
39      @opened_file.setter
40      def opened_file(self, value):
41          self._opened_file = value
42
        4 usages
43      @property
44      def schema_text(self):
45          return self._schema_text
46
        2 usages
47      @schema_text.setter
48      def schema_text(self, value):
49          self._schema_text = value
```

```
1   from tkinter import *
2   from tkinter import ttk
3
4   from app.services.json_schema_service import JSONSchemaService
5   from app.views.create_new_page import CreateNewPage
6   from app.views.strategy.I_view_strategy import IViewStrategy
7
8
    2 usages
9   class FirstPageViewStrategy(IViewStrategy):
10
11      # Створюємо об'єкт класу CreateNewPage()
12      create_new_window = CreateNewPage()
13      font = ('Consolas 20', 18)
14      width = 30
15
16      # Ініціалізуємо атрибути класу
17      def __init__(self):
18          super().__init__()
```

```
    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
```

```
from abc import ABC, abstractmethod


9 usages
class IViewStrategy(ABC):
    1 usage
    @abstractmethod
    def create_operating_window(self, window):
        pass
```

```
from app.views.strategy.I_view_strategy import IViewStrategy


5 usages
class ViewContext:
    strategy = IViewStrategy

    def __init__(self, strategy: IViewStrategy):
        super().__init__()
        self.strategy = strategy

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

```python
import tkinter as tk
from tkinter import *
import ctypes
from app.services.json_schema_service import JSONSchemaService
from app.views.create_new_page import CreateNewPage
from app.views.strategy.edit_page_view_strategy import EditPageViewStrategy


2 usages
class MenuView:
    # Створюємо об'єкт класу CreateNewPage()
    create_new_window = CreateNewPage()
    # Створюємо об'єкт класу JSONSchemaService()
    json_schema_service = JSONSchemaService()

    def __init__(self):
        super().__init__()
```

```python
    def create_menu(self, window):
        # Створюємо об'єкт меню
        self.menu = Menu()
        # Додаємо пункт меню "Файл"
        self.file_menu = Menu(self.menu, tearoff=0, font=("Helvetica", 16))
        self.menu.add_cascade(label="Файл", menu=self.file_menu, font=("Helvetica", 16))
        self.file_menu.add_command(label="Відкрити", font=('Helvetica', 20)
                                   , command=lambda: self.create_new_window.open_new_window(window))
        self.file_menu.add_command(label="Зберегти", font='10'
                                   , command=lambda: self.json_schema_service.
                                   save_file(EditPageViewStrategy.schema_text, EditPageViewStrategy.opened_file))
        self.file_menu.add_command(label="Зберегти як...", font='10'
                                   # , command=lambda: self.json_schema_service.save_as_file()
                                    )
        # self.protocol("WM_DELETE_WINDOW", self.on_closing)
        return self.menu
```

```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
class CreateNewPage:
    json_schema_service = JSONSchemaService()

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

    2 usages
    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

    def create_new_window(self):
        pass
```

```python
from tkinter import ttk
import tkinter as tk
from tkinter import *
import ctypes
from app.views.view_elements.menu_view import MenuView
from app.views.I_json_schema_view import IJSONSchemaView
from app.views.strategy.first_page_view_strategy import FirstPageViewStrategy
from app.views.strategy.view_context import ViewContext


# Створюємо клас для нашого додатку
2 usages  Valeriia Khyzhniak *
class JSONSchemaView(Tk, IJSONSchemaView):
    menu_view = MenuView()
    # Змінна для збереження стилю шрифту
    font = ('Consolas 20', 18)

    # Ініціалізуємо атрибути класу
    Valeriia Khyzhniak *
    def __init__(self):
        super().__init__()
        # Встановлюємо назву вікна
        self.title("JSON Schema Editor")
```

```python
        # Встановлюємо розмір вікна
        self.geometry("1200x600")
        # 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)
        # Створюємо об'єкт класу Style(), щоб налаштувати власний стиль
        s = ttk.Style()
        # Змінюємо розмір шрифту
        s.configure( style: 'TNotebook.Tab', font=self.font)
        # Створюємо контейнер, щоб можна було відображати декілька вкладок
        self.window = ttk.Notebook()
        # Розміщуємо об'єкт на головному вікні
        self.window.pack(expand=True)
        # Додаємо меню
        self.config(menu=self.menu_view.create_menu(self.window))
        # Створюємо початкову сторінку
        self.welcome_page = ViewContext(FirstPageViewStrategy()).view_page_strategy(self.window)
```

```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()
```

```python
    Valeriia Khyzhniak *
    def delete_all(self):
        with self.connection.cursor() as cursor:
            cursor.execute('DELETE FROM json_histories')
        self.connection.commit()

```

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

    new *
    def _insert(self, json_history):
        self.cursor.execute(
            "INSERT INTO json_histories (history_id, json_id, date_of_saving) VALUES (?, ?, ?)"
            , (json_history.history_id, json_history.json_id, json_history.date_of_saving))
        self.connection.commit()
```

```python
    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()
```

```python
    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.