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 №8

Технологія розробки програмного забезпечення

«ШАБЛОНИ «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR»»

Варіант 28

Виконала:
студентка групи ІА–13
Хижняк Валерія Валеріївна

Перевірив:
Мягкий М.Ю.

Київ 2023

Тема: Шаблони «Composite», «Flyweight», «Interpreter», «Visitor».

Мета: ознайомитись з шаблонами «Composite», «Flyweight», «Interpreter», «Visitor», 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:

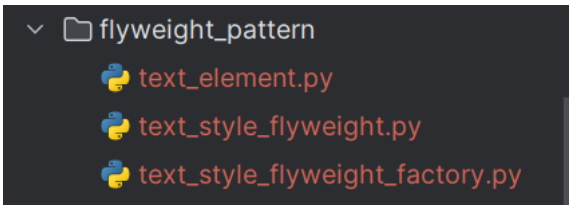
..28 JSON Tool (ENG) (strategy, command, observer, template method, flyweight)

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 Flyweight.

Суть патерну Flyweight полягає в тому, щоб спільно використовувати частину стану об'єктів та уникати дублювання інформації, яка може бути загальною для багатьох об'єктів. Замість того, щоб зберігати інформацію в кожному об'єкті окремо, цю інформацію виокремлюють в окремий об'єкт (Flyweight), який може бути використаний багатьма об'єктами.

Даний патерн реалізований в дерикторії flyweight_pattern:



Клас TextStyleFlyweightFactory це клас, який відповідальний за створення і зберігання об’єктів класу TextStyleFlyweight замість створення нових екземплярів для кожного виклику. В цьому класі міститься метод get_style який

отримує в якості параметрів властивості стилю і перевіряє чи заданий стиль є в списку стилів. Якщо такий стиль відсутній, то створює новий екземпляр з властивостями цього стилю, якщо наявний – бере з масиву стилів, та повертає цей екземпляр.

```python
from app.views.flyweight_pattern.text_style_flyweight import TextStyleFlyweight


2 usages
class TextStyleFlyweightFactory:
    # Змінна, яка зберігає в собі список наявних стилів
    _styles = {}

    1 usage
    def get_style(self, font_family, font_size, text_color):
        # Створюємо кортеж, що визначає унікальний ключ для ідентифікації
        # конкретного стилю тексту на основі його властивостей
        key = (font_family, font_size, text_color)
        # Перевіряємо чи стиль з вказаними властивостями вже існує в колекції
        if key not in self._styles:
            # Якщо ні, то створюємо новий екземпляр та додаємо його до списку
            self._styles[key] = TextStyleFlyweight(font_family, font_size, text_color)
        return self._styles[key]
```

TextStyleFlyweight це клас, який ініціалізує об'єкт текстового стилю з вказаними властивостями: font_family, font_size і text_color. Створення екземпляру даного класу відбувається в класі TextStyleFlyweightFactory.

```python
2 usages
class TextStyleFlyweight:
    def __init__(self, font_family, font_size, text_color):
        self.font_family = font_family
        self.font_size = font_size
        self.text_color = text_color
```

TextElement це клас, який представляє елемент тексту. Він приймає параметри master (батьківський віджет, до якого буде додано Label), text (текст для відображення) та style (об'єкт стилю тексту). В конструкторі створюється віджет з текстом. Далі в методі apply_style конфігурується віджет, встановлюючи властивості шрифту та кольору тексту згідно із заданим стилем. Метод display

містить виклик методу apply_style та запаковує створений віджет для відображення.

```python
from tkinter import *


class TextElement:
    def __init__(self, master, text, style):
        self.label = Label(master, text=text)
        self.style = style

    1 usage
    def apply_style(self):
        self.label.config(
            font=(self.style.font_family, self.style.font_size),
            fg=self.style.text_color
        )

    def display(self):
        self.apply_style()
        self.label.pack()
```

Створення стилю відбувається в класі JSONSchemaView. Тут створюється фабрика та для неї викликається метод get_style, результати виконання якого записуються в змінну _common_style.

```python
2 usages  Valeriia Khyzhniak *
class JSONSchemaView(Tk, IJSONSchemaView):
    menu_view = MenuView()
    # Створюємо фабрику
    text_style_factory = TextStyleFlyweightFactory()
    # Створюємо загальний стиль для тексту
    _common_style = text_style_factory.get_style( font_family: "Consolas",  font_size: 20,  text_color: "black")
```

Створений стиль використовується для зміни стилю тексту вкладок.

```python
# Змінюємо стиль тексту (в параметр font передаємо значення з загального стилю)
s.configure( style: 'TNotebook.Tab', font=(self._common_style.font_family, self._common_style.font_size))
```

Висновок: на даній лабораторній роботі я ознайомилась з шаблонами «Composite», «Flyweight», «Interpreter», «Visitor», реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей, застосувала один з шаблонів при реалізації програми.