МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №9

Створення та рефакторінг програмно-інформаційного продукту засобами Python

Виконала:

ст. гр. РІ-32
Довгошия А.А

Балів	**Дата**

Прийняв:

асис. каф. ІСМ

Щербак С.С

Львів — 2024

Мета: розробка програмно-інформаційного продукту засобами Python

Хід роботи

Завдання 1. Створити скрипт запуску лабораторних робіт 1-8 (Runner) з єдиним меню для управління додатками використовуючи патерн FACADE https://refactoring.guru/uk/design-patterns/facade

Завдання 2. Зробити рефакторінг додатків, які були зроблені в лб 1-8, для підтримки можливості запуску через Runner

Завдання 3. Зробити рефакторинг додатків, які були зроблені в лб 1-8, використовуючи багаторівневу архітектуру додатків (див. приклад нижче) та всі принципи об’єктно-орієнтованого підходу

Завдання 4. Створити бібліотеку класів, які повторно використовуються у всіх лабораторних роботах та зробити рефакторінг додатків для підтримки цієї бібліотеки. Таких класів в бібліотеці має буде як найменш 5

Завдання 5. Додати логування функцій в класи бібліотеки програмного продукту використовуючи https://docs.python.org/uk/3/howto/logging.html

Завдання 6. Додати коментарі до програмного коду та сформувати документацію програмного продукту засобами pydoc. Документація має бути представлена у вигляді сторінок тексту на консолі, подана у веб-браузері та збережена у файлах HTML

Завдання 7. Документація та код програмного продукту має бути розміщено в GIT repo

Завдання 8. Проведіть статичний аналіз коду продукту засобами PYLINT https://pylint.readthedocs.io/en/stable/ та виправте помилки, які були

ідентифіковані. Первинний репорт з помилками додайте до звіту лабораторної роботи

Завдання 9. Підготуйте звіт до лабораторной роботи

Lib (бібліотека класів)

Api_service.py

```
import requests
from src.const.const import BASE_URL

class ApiService:
    """Handles API requests and responses."""
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

    def get_data(self, endpoint):
        """Fetches data from the specified API endpoint."""
        try:
            response = requests.get(f"{BASE_URL}/{endpoint}")
            response.raise_for_status()
            return response.json()
        except requests.exceptions.HTTPError as http_err:
            print(f"HTTP error occurred: {http_err}")
```

```
            return None
        except Exception as err:
            print(f"An error occurred: {err}")
            return None
```

data_saver.py

```
import os
import json
import csv
from src.const.path_constants import FOLDER_PATH_RESULT_LAB7

class DataSaver:
    """Handles the saving of data in different formats."""
    @staticmethod
    def save_data(data, format_choice, filename):
        """Saves the data to a specified format and filename."""
        results_dir = FOLDER_PATH_RESULT_LAB7
        os.makedirs(results_dir, exist_ok=True)

        file_path = os.path.join(results_dir, filename)

        if format_choice == 'json':
            DataSaver.save_json(data, file_path)
        elif format_choice == 'csv':
            DataSaver.save_csv(data, file_path)
        elif format_choice == 'txt':
```

```
            DataSaver.save_txt(data, file_path)
        else:
            print("Unsupported format type.")

    @staticmethod
    def save_json(data, file_path):
        """Saves data in JSON format."""
        with open(file_path, 'w') as json_file:
            json.dump(data, json_file, indent=4)
        print(f"Data saved in JSON format to {file_path}")

    @staticmethod
    def save_csv(data, file_path):
        """Saves data in CSV format."""
        with open(file_path, 'w', newline='') as csv_file:
            writer = csv.writer(csv_file)
            if data:
                writer.writerow(data[0].keys())
                for item in data:
                    writer.writerow(item.values())
        print(f"Data saved in CSV format to {file_path}")

    @staticmethod
    def save_txt(data, file_path):
        """Saves data in plain text format."""
        with open(file_path, 'w') as txt_file:
```

```
            for item in data:
                txt_file.write(str(item) + "\n")
        print(f"Data saved in plain text format to {file_path}")
```

file_operations.py

```
import os

class FileOperations:
    @staticmethod
    def save_art(ascii_art, base_path, filename):
        ascii_artworks_path = os.path.join(base_path, "ascii_artworks")

        # Ensure the ascii_artworks directory exists
        os.makedirs(ascii_artworks_path, exist_ok=True)

        # Define the full file path for saving
        file_path = os.path.join(ascii_artworks_path, filename)

        try:
            with open(file_path, 'w') as file:
                file.write(ascii_art)
            print(f"~ saved in  {file_path}\n")
        except Exception as e:
            print(f"! an error occurred while saving: {e}")
```

history_logger.py

```
from datetime import datetime

class HistoryLogger:
    """Logs the history of user choices."""
    @staticmethod
    def log(user_choice):
        """Logs the user's choice."""
        with open('user_history.log', 'a') as log_file:
            log_file.write(f"{datetime.now()} - {user_choice}\n")
```

user_input.py

```
class UserInput:
    @staticmethod
    def get_input(prompt, choices=None):
        while True:
            value = input(prompt).strip().lower()
            if not value:
                print(f"\t! invalid value")
                continue
            if choices and value not in choices:
                formatted_choices = ", ".join(choices)
                print(f"\t! incorrect choice. options: {formatted_choices}")
            else:
                return value
```

```
    @staticmethod
    def get_integer_input(prompt, min_value, max_value):
        while True:
            value = input(prompt).strip()
            if not value:
                print(f"\t! invalid value")
                continue
            try:
                value = int(value)
                if min_value <= value <= max_value:
                    return value
                else:
                    print(f"\t! value is out of a range ({min_value} - {max_value})")
            except ValueError:
                print(f"\t! invalid value")
```

runner.py

```
from src.classes.lab1 import main as lab1
from src.classes.lab2 import main as lab2
from src.classes.lab3 import main as lab3
from src.classes.lab4 import main as lab4
from src.classes.lab5 import main as lab5
from src.classes.lab7 import main as lab7
from src.classes.lab8 import main as lab8
```

```
def display_menu():
    print("\nChoose lab:\n"
        "\t1. Lab1\n"
        "\t2. Lab2\n"
        "\t3. Lab3\n"
        "\t4. Lab4\n"
        "\t5. Lab5\n"
        "\t7. Lab7\n"
        "\t8. Lab8\n"
        "\t0. exit\n")


def main():
    options = {
        '1': lab1.main,
        '2': lab2.main,
        '3': lab3.main,
        '4': lab4.main,
        '5': lab5.main,
        '7': lab7.main,
        '8': lab8.main,
        '0': lambda: print("Exiting program...")
    }
```

```
    while True:
        print("-----------")
        display_menu()
        user_input = input('Enter number: ')

        action = options.get(user_input, lambda: print("Invalid option. Please
choose again."))
        action()

        if user_input == '0':
            break


if __name__ == "__main__":
    main()
```

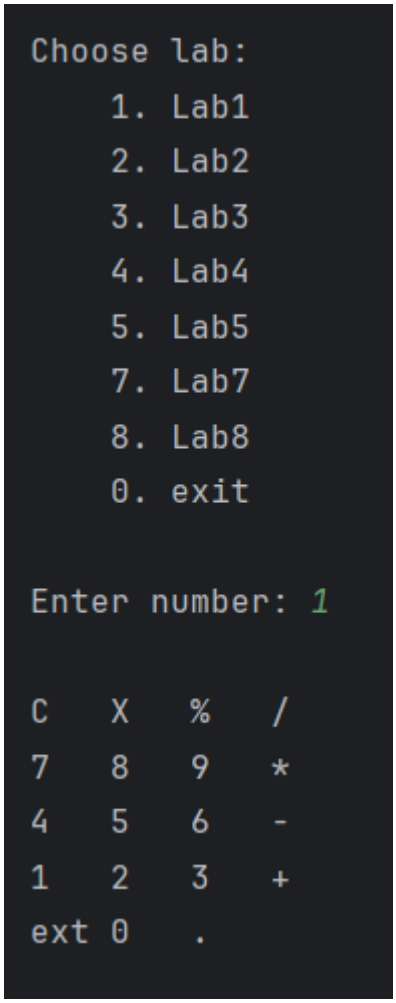

Рис.1. Результат програми.

Висновок: на цій лабораторній роботі я вивчила розробку програмно-інформаційного продукту засобами Python