МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж



Лабораторна робота №9

з дисципліни «Спеціалізовані мови програмування»

на тему «Створення та рефакторінг програмно-інформаційного продукту засобами Python»

Виконала студентка

групи РІ-32

Титиш О.І.

Прийняв:

Щербак С.С

Львів – 2024

**Мета:** розробка програмно-інформаційного продукту засобами Python.

**План 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

**Код програми**

Library/console\_menu

from library.menu\_interface import MenuInterface  
from library.input\_validator import InputValidator  
  
  
class ConsoleMenu(MenuInterface):  
 *"""Реалізація консольного меню"""* def \_\_init\_\_(self, title, subtitle):  
 self.title = title  
 self.subtitle = subtitle  
 self.items = {}  
 self.running = True  
  
 def add\_item(self, name, function):  
 *"""Додавання пункту меню з назвою та пов’язаною функцією"""* next\_index = len(self.items) + 1  
 self.items[next\_index] = (name, function)  
  
 def show(self):  
 *"""Відображення меню та обробка введених даних користувача"""* while self.running:  
 self.\_display\_menu()  
 self.\_handle\_choice()  
  
 def \_display\_menu(self):  
 *"""Відображення пунктів меню"""* print(f"\n{'=' \* 50}")  
 print(f"{self.title.center(50)}")  
 print(f"{self.subtitle.center(50)}")  
 print(f"{'=' \* 50}\n")  
  
 for index, (name, \_) in self.items.items():  
 print(f"{index}. {name}")  
 print("\n0. Exit")  
  
 def \_handle\_choice(self):  
 *"""Керування вибором меню користувача"""* choice = InputValidator.validate\_number\_input(  
 "\nEnter your choice: ",  
 min\_value=0,  
 max\_value=len(self.items)  
 )  
  
 if choice == 0:  
 self.running = False  
 print("\nGoodbye!")  
 else:  
 try:  
 print(f"\nExecuting: {self.items[choice][0]}")  
 self.items[choice][1]()  
 except Exception as e:  
 print(f"Error executing command: {str(e)}")  
 input("\nPress Enter to continue...")

Library/façade

from library.logger import Logger  
  
  
class LabFacade:  
 *"""Реалізація патерну Facade"""* def \_\_init\_\_(self):  
 self.logger = Logger()  
  
 @Logger().log\_function\_call  
 def execute\_lab1(self):  
 *"""Виконати Lab 1"""* from lab1.main\_LAB1 import run as run\_lab1  
 run\_lab1()  
  
 @Logger().log\_function\_call  
 def execute\_lab2(self):  
 *"""Виконати Lab 2"""* from lab2.main\_LAB2 import calculator  
 calculator()  
  
 @Logger().log\_function\_call  
 def execute\_lab3(self):  
 *"""Виконати Lab 3"""* from lab3.main\_LAB3 import start\_console\_interface  
 start\_console\_interface()  
  
 @Logger().log\_function\_call  
 def execute\_lab4(self):  
 *"""Виконати Lab 4"""* from lab4.main\_LAB4 import run as run\_lab4  
 run\_lab4()  
  
 @Logger().log\_function\_call  
 def execute\_lab5(self):  
 *"""Виконати Lab 5"""* from lab5.main\_LAB5 import main  
 main()  
  
 @Logger().log\_function\_call  
 def execute\_lab6(self):  
 *"""Виконати Lab 6"""* import unittest  
 from lab6.test\_calculator\_LAB6 import TestCalculator  
 suite = unittest.TestLoader().loadTestsFromTestCase(TestCalculator)  
 runner = unittest.TextTestRunner()  
 runner.run(suite)  
  
 @Logger().log\_function\_call  
 def execute\_lab7(self):  
 *"""Виконати Lab 7"""* from lab7.main\_LAB7 import run as run\_lab7  
 run\_lab7()  
  
 @Logger().log\_function\_call  
 def execute\_lab8(self):  
 *"""Виконати Lab 8"""* from lab8.main\_LAB8 import run as run\_lab8  
 run\_lab8()

Library/input\_validator

import logging  
import logging.config  
import json  
from functools import wraps  
  
class InputValidator:  
 *"""Клас для перевірки вхідних даних"""* @staticmethod  
 def validate\_number\_input(prompt, min\_value=None, max\_value=None):  
 *"""Перевірте числове введення в необов’язковому діапазоні"""* while True:  
 try:  
 value = int(input(prompt))  
 if min\_value is not None and value < min\_value:  
 print(f"Value must be at least {min\_value}")  
 continue  
 if max\_value is not None and value > max\_value:  
 print(f"Value must be at most {max\_value}")  
 continue  
 return value  
 except ValueError:  
 print("Please enter a valid number")  
  
 @staticmethod  
 def validate\_string\_input(prompt, min\_length=None, max\_length=None):  
 *"""Перевірте введений рядок із необов’язковими обмеженнями довжини"""* while True:  
 value = input(prompt).strip()  
 if min\_length is not None and len(value) < min\_length:  
 print(f"Input must be at least {min\_length} characters long")  
 continue  
 if max\_length is not None and len(value) > max\_length:  
 print(f"Input must be at most {max\_length} characters long")  
 continue  
 return value

Library/logger

import logging  
import logging.config  
import json  
from functools import wraps  
  
  
class Logger:  
 *"""Клас для обробки операцій логування"""* \_instance = None  
  
 def \_\_new\_\_(cls):  
 if cls.\_instance is None:  
 cls.\_instance = super(Logger, cls).\_\_new\_\_(cls)  
 try:  
 with open('config/logging\_config.json', 'r', encoding='utf-8') as config\_file:  
 config = json.load(config\_file)  
 logging.config.dictConfig(config)  
 except FileNotFoundError:  
 logging.basicConfig(level=logging.INFO)  
 cls.\_instance.logger = logging.getLogger(\_\_name\_\_)  
 return cls.\_instance  
  
 def log\_function\_call(self, func):  
 *"""Декоратор для логування викликів функцій"""* @wraps(func)  
 def wrapper(\*args, \*\*kwargs):  
 self.logger.info(f"Starting: {func.\_\_name\_\_}")  
 try:  
 result = func(\*args, \*\*kwargs)  
 self.logger.info(f"Completed: {func.\_\_name\_\_}")  
 return result  
 except Exception as e:  
 self.logger.error(f"Error in {func.\_\_name\_\_}: {str(e)}")  
 raise  
  
 return wrapper

Library/menu\_interface

from abc import ABC, abstractmethod  
  
  
class MenuInterface(ABC):  
 *"""Абстрактний базовий клас для реалізації меню"""* @abstractmethod  
 def show(self):  
 *"""Відобразити меню"""* pass  
  
 @abstractmethod  
 def add\_item(self, name, function):  
 *"""Додати пункт до меню"""* pass

Runner.py

from library.console\_menu import ConsoleMenu  
from library.facade import LabFacade  
from library.logger import Logger  
  
  
class Runner:  
 *"""  
 Основний клас runner, який забезпечує уніфікований інтерфейс для виконання всіх лабораторних робіт.  
 Реалізує шаблон facade, щоб спростити інтерфейс для запуску різних лабораторних робіт.  
 """* def \_\_init\_\_(self):  
 *"""Ініціалізація основного класу з меню та патерном"""* self.logger = Logger()  
 self.facade = LabFacade()  
 self.menu = ConsoleMenu(  
 "Laboratory Works Runner",  
 "Please select a lab to run"  
 )  
 self.\_setup\_menu()  
  
 def \_setup\_menu(self):  
 *"""Налаштуйте пункти меню з відповідними функціями"""* self.menu.add\_item("Lab 1: Simple Operations", self.facade.execute\_lab1)  
 self.menu.add\_item("Lab 2: Calculator", self.facade.execute\_lab2)  
 self.menu.add\_item("Lab 3: Console Interface", self.facade.execute\_lab3)  
 self.menu.add\_item("Lab 4: ASCII Art", self.facade.execute\_lab4)  
 self.menu.add\_item("Lab 5: 3D Shapes", self.facade.execute\_lab5)  
 self.menu.add\_item("Lab 6: Unit Tests", self.facade.execute\_lab6)  
 self.menu.add\_item("Lab 7: User Repository", self.facade.execute\_lab7)  
 self.menu.add\_item("Lab 8: Data Visualization", self.facade.execute\_lab8)  
  
 @Logger().log\_function\_call  
 def run(self):  
 *"""Запустіть runner і покажіть меню"""* try:  
 self.menu.show()  
 except KeyboardInterrupt:  
 print("\nExiting the program...")  
 except Exception as e:  
 print(f"An error occurred: {str(e)}")  
 raise

if \_\_name\_\_ == "\_\_main\_\_":  
 runner = Runner()  
 runner.run()

**Результати виконання програми**

Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

*Рис. 1. Результат виконання*

**Висновок:** Виконуючи лабораторну роботу було створено програмно-інформаційний продукт засобами Python. Цей проект покращив мої навички створення архітектурних проектів.