# **МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА” ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ** **Кафедра ІСМ**

# **Звіт**

# **до лабораторної роботи №9**

# **З дисципліни “Спеціальні мови програмування”**

**Виконав:  
ст.гр.ІТ-31  
Шельвах Максим  
  
Прийняв:  
Щербак С.С.**

# **Львів — 2023**

**Тема роботи:** Створення та рефакторінг програмно-інформаційного продукту засобами Python.

**Мета роботи:** 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 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

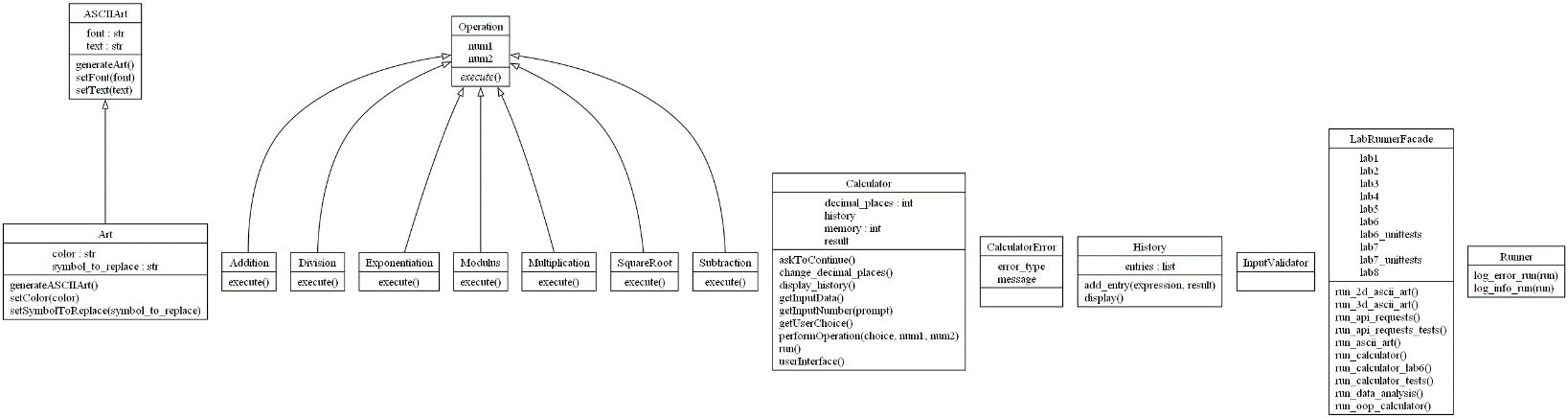
Завдання 7. Документація та код програмного продукту має бути розміщено в GIT repo

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.

Хід роботи

Діаграма класів реалізованих у розробленій програмі та папці classes:



**Реалізований код:**

.\classes\facade\facade.py

class LabRunnerFacade:

"""

A facade class for running various laboratory applications.

This class acts as a central point for executing different lab tasks,

encapsulating the complexity of individual lab module executions.

It imports and manages several lab modules, each representing a different

task such as calculators, ASCII art creators, and data analysis tools.

Attributes:

lab1 (module): Module for running the basic calculator from lab1.

lab2 (module): Module for running the OOP calculator from lab2.

lab3 (module): Module for running the ASCII art generator from lab3.

lab4 (module): Module for running the 2D ASCII art generator from lab4.

lab5 (module): Module for running the 3D ASCII art generator from lab5.

lab6 (module): Module for running the calculator from lab6.

lab6\_unittests (module): Module for running unit tests for lab6 calculator.

lab7 (module): Module for running API requests from lab7.

lab7\_unittests (module): Module for running unit tests for lab7 API requests.

lab8 (module): Module for running data analysis from lab8.

"""

def \_\_init\_\_(self):

"""

Initializes the LabRunnerFacade instance.

Sets up logging and imports necessary lab modules, storing their references

for later use. Each lab module is accessible as an attribute of the instance.

"""

# Import modules here

import core.lab1.main

import core.lab2.main

import core.lab3.main

import core.lab4.main

import core.lab5.main

import core.lab6.main

import core.lab6.unit\_test

import core.lab7.main

import core.lab7.unittests

import core.lab8.main

# Store module references in the facade

self.lab1 = core.lab1.main

self.lab2 = core.lab2.main

self.lab3 = core.lab3.main

self.lab4 = core.lab4.main

self.lab5 = core.lab5.main

self.lab6 = core.lab6.main

self.lab6\_unittests = core.lab6.unit\_test

self.lab7 = core.lab7.main

self.lab7\_unittests = core.lab7.unittests

self.lab8 = core.lab8.main

def run\_calculator(self):

"""

Runs the basic calculator application from lab1.

"""

self.lab1.main()

def run\_oop\_calculator(self):

"""

Runs the Object-Oriented Programming (OOP) calculator from lab2.

"""

self.lab2.main()

def run\_ascii\_art(self):

"""

Executes the ASCII art generator from lab3.

"""

self.lab3.main()

def run\_2d\_ascii\_art(self):

"""

Executes the 2D ASCII art generator from lab4.

"""

self.lab4.main()

def run\_3d\_ascii\_art(self):

"""

Executes the 3D ASCII art generator from lab5.

"""

self.lab5.main()

def run\_calculator\_lab6(self):

"""

Runs the advanced calculator application from lab6.

"""

self.lab6.main()

def run\_calculator\_tests(self):

"""

Executes the unit tests for the calculator application from lab6.

"""

self.lab6\_unittests.main()

def run\_api\_requests(self):

"""

Executes the API requests application from lab7.

"""

self.lab7.main()

def run\_api\_requests\_tests(self):

"""

Executes the unit tests for the API requests application from lab7.

"""

self.lab7\_unittests.main()

def run\_data\_analysis(self):

"""

Runs the data analysis application from lab8.

"""

self.lab8.main()

runner.py

"""

This module implements a console menu for launching various programs.

It utilizes ConsoleMenu to create a user interface.

"""

import logging

from consolemenu import ConsoleMenu

from consolemenu.items import FunctionItem

from classes.facade.facade import LabRunnerFacade

class Runner(LabRunnerFacade):

def \_\_init\_\_(self):

"""

Initializes the Runner instance.

Sets up the logging configuration and initializes the LabRunnerFacade.

"""

super().\_\_init\_\_()

logging.basicConfig(filename='logs/example.log', level=logging.DEBUG)

def log\_info\_run(self, run):

"""

Logs information about the program being run.

Args:

run (str): Description of the program being run.

"""

logging.info('Starting: %s', run)

def log\_error\_run(self, run):

"""

Logs uncaught errors that occur during program execution.

Args:

run (str): Description of the program where the error occurred.

"""

logging.error('Failed to execute: %s', run)

def main():

"""

Main function to initialize and display the console menu.

It initializes the console menu with a title 'Runner', sets up the Runner object,

and adds various menu items that trigger methods of the Runner object. In case of any

exceptions, it logs the error.

"""

try:

runner = Runner()

menu = ConsoleMenu("Runner")

runner.log\_info\_run("ConsoleMenu initialized")

runner.log\_info\_run("Runner initialized")

menu.append\_item(FunctionItem("Calculator", runner.run\_calculator))

menu.append\_item(FunctionItem("OOP Calculator", runner.run\_oop\_calculator))

menu.append\_item(FunctionItem("ASCII Art", runner.run\_ascii\_art))

menu.append\_item(FunctionItem("2D ASCII Art without additional libraries", runner.run\_2d\_ascii\_art))

menu.append\_item(FunctionItem("3D ASCII Arts", runner.run\_3d\_ascii\_art))

menu.append\_item(FunctionItem("Calculator (Lab6)", runner.run\_calculator\_lab6))

menu.append\_item(FunctionItem("Calculator tests", runner.run\_calculator\_tests))

menu.append\_item(FunctionItem("API requests", runner.run\_api\_requests))

menu.append\_item(FunctionItem("API requests tests", runner.run\_api\_requests\_tests))

menu.append\_item(FunctionItem("Data analysis", runner.run\_data\_analysis))

menu.show()

runner.log\_info\_run("Menu displayed")

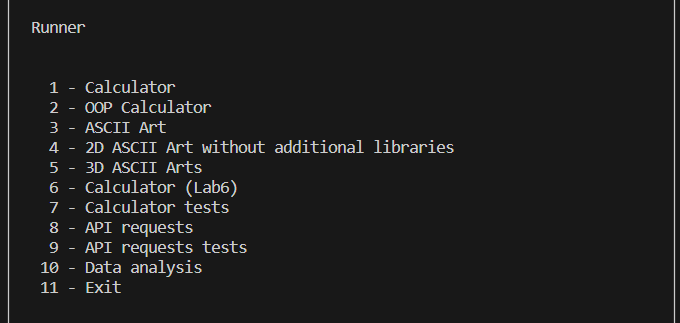
except Exception as e:

runner.log\_error\_run(f"An error occurred: {str(e)}")

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Результат виконання програми:**



**Висновок:** виконавши ці завдання, я створив програмно-інформаційний продукт засобами Python на основі класів з попередніх лабораторних робіт..