Міністерство освіти і науки України  
Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра інформаційних систем та мереж

******

**ЗВІТ**

**ДО ЛАБОРАТОРНОЇ РОБОТИ №2  
з дисципліни  
«Спеціалізовані мови програмування»  
на тему:**  
«Основи побудови об’єктно-орієнтованих додатків на Python»

***Виконала:*** *студентка групи РІ-31  
 Діана ВІЙЧУК*

***Прийняв:****к. т. н., доцент* *Сергій ЩЕРБАК*

## МЕТА РОБОТИ

Розробка консольного калькулятора в об’єктно орієнтованому стилі з використанням класів.

## ЗАВДАННЯ

Завдання 1: Створення класу Calculator

Створіть клас Calculator, який буде служити основою для додатка калькулятора.

Завдання 2: Ініціалізація калькулятора

Реалізуйте метод \_\_init\_\_ у класі Calculator для ініціалізації необхідних атрибутів або змінних.

Завдання 3: Введення користувача

Перемістіть функціональність введення користувача в метод у межах класу Calculator. Метод повинен приймати введення для двох чисел і оператора.

Завдання 4: Перевірка оператора

Реалізуйте метод у класі Calculator, щоб перевірити, чи введений оператор є дійсним (тобто одним із +, -, \*, /). Відобразіть повідомлення про помилку, якщо він не є дійсним.

Завдання 5: Обчислення

Створіть метод у класі Calculator, який виконує обчислення на основі введення користувача (наприклад, додавання, віднімання, множення, ділення).

Завдання 6: Обробка помилок

Реалізуйте обробку помилок у межах класу Calculator для обробки ділення на нуль або інших потенційних помилок. Відобразіть відповідні повідомлення про помилку.

Завдання 7: Повторення обчислень

Додайте метод до класу Calculator, щоб запитати користувача, чи він хоче виконати ще одне обчислення. Якщо так, дозвольте йому ввести нові числа і оператор. Якщо ні, вийдіть з програми.

Завдання 8: Десяткові числа

Модифікуйте клас Calculator для обробки десяткових чисел (плаваюча кома) для більш точних обчислень.

Завдання 9: Додаткові операції

Розширте клас Calculator, щоб підтримувати додаткові операції, такі як піднесення до степеня (^), квадратний корінь (√) та залишок від ділення (%).

Завдання 10: Інтерфейс, зрозумілий для користувача

Покращте інтерфейс користувача у межах класу Calculator, надавши чіткі запити, повідомлення та форматування виводу для зручності читання.

## ПРОГРАМНА РЕАЛІЗАЦІЯ

Файл *runner.py* .

import sys

import os

lab2\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", ".."))

sys.path.append(lab2\_root)

from src.main import main

if \_\_name\_\_ == "\_\_main\_\_":

main()

Файл */src/bll/classes/AdvancedCalculator.py* .

import math

from src.bll.classes.BaseCalculator import BaseCalculator

class AdvancedCalculator(BaseCalculator):

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

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

def calculate(self, operand1, operator, operand2=None):

if operator == 'log':

if operand1 <= 0:

print("Логарифм може бути обчислений лише для додатних чисел.")

return None

return math.log(operand1)

elif operator == 'sin':

return math.sin(math.radians(operand1))

else:

return super().calculate(operand1, operator, operand2)

def run(self):

print("Запущено розширений калькулятор")

super().run()

Файл */src/bll/classes/BaseCalculator.py* .

from abc import ABC, abstractmethod

from shered.dal.history import show\_history, add\_to\_history

class BaseCalculator(ABC):

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

self.\_\_decimal\_places = 2

self.memory\_value = None

@abstractmethod

def calculate(self, operand1, operator, operand2=None):

pass

def update\_settings(self, decimal\_places):

self.\_\_decimal\_places = decimal\_places

print(f"Налаштування успішно змінено: {decimal\_places} знаків після коми.")

def display\_result(self, result):

print(f"Результат: {result:.{self.\_\_decimal\_places}f}")

def run(self):

while True:

action = input("Виберіть дію: 1. Обчислити, 2. Змінити налаштування, 3. Вийти: ")

if action == '1':

operands = self.get\_user\_input()

result = self.calculate(\*operands)

if result is not None:

self.display\_result(result)

self.save\_to\_memory(result)

elif action == '2':

self.change\_settings()

elif action == '3':

if input("Вивести історію? (y/n): ").lower() == 'y':

self.show\_history()

break

else:

print("Невірний вибір. Спробуйте ще раз.")

def get\_user\_input(self):

use\_memory = input("Використовувати число з пам'яті? (y/n): ").lower() == 'y'

if use\_memory:

if self.memory\_value is not None:

print(f"Число з пам'яті: {self.memory\_value}")

operand1 = self.memory\_value

else:

print("Немає збереженого значення в пам'яті.")

operand1 = float(input("Введіть перше число: "))

else:

operand1 = float(input("Введіть перше число: "))

operator = input("Введіть оператор (+, -, \*, /, ^, sq, log, sin): ")

operand2 = None

if operator not in ['sq', 'log', 'sin']:

operand2 = float(input("Введіть друге число: "))

return operand1, operator, operand2

def change\_settings(self):

new\_decimal\_places = int(input("Введіть кількість знаків після коми: "))

self.update\_settings(new\_decimal\_places)

def save\_to\_memory(self, result):

self.memory\_value = round(result, self.\_\_decimal\_places)

print(f"Збережено {self.memory\_value} в пам'яті.")

def show\_history(self):

show\_history()

Файл */src/bll/classes/Calculator.py* .

import sys

import os

lab2\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", ".."))

sys.path.append(lab2\_root)

from src.bll.classes.BaseCalculator import BaseCalculator

from bll.operations import Addition, Subtraction, Division, Multiplication, Power, SquareRoot, FindRemainder

from shered.dal.memory import save\_result, get\_result, has\_memory

from shered.dal.history import add\_to\_history, show\_history

class Calculator(BaseCalculator):

def \_\_init\_\_(self, operand1=None, operand2=None, operator=None):

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

self.operand1 = operand1

self.operand2 = operand2

self.operator = operator

self.result = None

def get\_operation(self, operator):

if operator == '+':

return Addition()

elif operator == '-':

return Subtraction()

elif operator == '\*':

return Multiplication()

elif operator == '/':

return Division()

elif operator == '^':

return Power()

elif operator == 'sq':

return SquareRoot()

elif operator == '%':

return FindRemainder()

else:

raise ValueError("Невідома операція")

def calculate(self, operand1, operator, operand2=None):

if not isinstance(operand1, (int, float)) or (operand2 is not None and not isinstance(operand2, (int, float))):

raise TypeError("Операнди повинні бути числами.")

try:

operation = self.get\_operation(operator)

except ValueError as e:

print(e)

return None

result = operation.execute(operand1, operand2)

if result is None:

print(f"Помилка: Оператор '{operator}' не підтримується.")

return None

add\_to\_history(operand1, operator, operand2, result)

return result

def run(self):

super().run()

Файл */src/main.py* .

import sys

import os

lab2\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "..", ".."))

sys.path.append(lab2\_root)

from src.bll.classes.Calculator import Calculator

from src.bll.classes.AdvancedCalculator import AdvancedCalculator

def main():

while True:

calculator\_type = input("Виберіть тип калькулятора (1 - Основний, 2 - Науковий): ")

if calculator\_type == '1':

calculator = Calculator()

break

elif calculator\_type == '2':

calculator = AdvancedCalculator()

break

else:

print("Невірний вибір. Спробуйте ще раз.")

calculator.run()

if \_\_name\_\_ == "\_\_main\_\_":

main()

## ВИСНОВКИ

На цій лабораторній роботі я розробила консольний калькулятор в об’єктно орієнтованому стилі з використанням класів.

Посилання на GitHub: <https://github.com/Dinasi4ka/Calculator.git>