Московский государственный технический университет им. Н. Э. Баумана
(МГТУ им. Н. Э. Баумана)

Кафедра «Системы обработки информации и управления» (ИУ5)

Лабораторная работа №5

По дисциплине: «Парадигмы и конструкции языков программирования»

«Шаблоны проектирования и модульное тестирование в Python»

Выполнила: Холодова К. А.,
студентка группы ИУ5Ц-52Б

Проверил: Гапанюк Ю. Е.

г. Москва 2023 г.

Цель лабораторной работы: изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python

Задание:

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3–4.
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:
   - TDD - фреймворк (не менее 3 тестов).
   - BDD - фреймворк (не менее 3 тестов).
   - Создание Mock-объектов (необязательное дополнительное задание).

Код программы

Файл main.py

```python
import math

def solving(a, b, c):
    D = b**2 - 4*a*c

    if D > 0:
        x1 = (-b + math.sqrt(D)) / (2*a)
        x2 = (-b - math.sqrt(D)) / (2*a)
        return x1, x2
    elif D == 0:
        x = -b / (2*a)
        return x
    else:
        return "No real roots"
```

Файл test_main_TDD.py

```python
import unittest
from main import solving

class TestSolving(unittest.TestCase):

    def test_positive_discriminant(self):
        self.assertEqual(solving(1, -3, 2), (2, 1))
        self.assertEqual(solving(2, -5, -3), (3, -0.5))
        self.assertEqual(solving(1, 0, -9), (3, -3))

    def test_zero_discriminant(self):
        self.assertEqual(solving(1, 4, 4), -2)
```

```
        self.assertEqual(solving(1, -2, 1), 1)
        self.assertEqual(solving(1, 6, 9), -3)

    def test_negative_discriminant(self):
        self.assertEqual(solving(1, 2, 5), "No real roots")
        self.assertEqual(solving(1, 0, 1), "No real roots")

if __name__ == '__main__':
    unittest.main()
```

Файл test_main_BDD.py

```
from behave import given, when, then
from main import solving

@given('coefficients a={a}, b={b}, c={c}')
def set_coefficients(context, a, b, c):
    context.coefficients = (int(a), int(b), int(c))

@when('I solve the quadratic equation')
def solve_quadratic(context):
    context.result = solving(*context.coefficients)

@then('the solutions should be {x1} and {x2}')
def check_solutions(context, x1, x2):
    expected_result = (float(x1), float(x2))
    assert context.result == expected_result, f"Expected {expected_result}, but
got {context.result}"

@then('the solution should be {x}')
def check_solution(context, x):
    expected_result = float(x)
    assert context.result == expected_result, f"Expected {expected_result}, but
got {context.result}"

@then('there should be no real roots')
def check_no_real_roots(context):
    assert context.result == "No real roots", f"Expected 'No real roots', but
got {context.result}"
```

Файл solving.feature

Feature: Solving quadratic equations

Scenario: Quadratic equation with two real roots

Given coefficients a=1, b=-3, c=2

When I solve the quadratic equation

Then the solutions should be 2 and 1

Scenario: Quadratic equation with one real root
Given coefficients a=1, b=-2, c=1
When I solve the quadratic equation
Then the solution should be 1

Scenario: Quadratic equation with no real roots
Given coefficients a=1, b=2, c=5
When I solve the quadratic equation
Then there should be no real roots

Scenario: Quadratic equation with positive discriminant
Given coefficients a=1, b=-1, c=-6
When I solve the quadratic equation
Then the solutions should be 3 and -2

Scenario: Quadratic equation with fractional roots
Given coefficients a=2, b=7, c=3
When I solve the quadratic equation
Then the solutions should be -0.5 and -3

Результат выполнения:

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```