**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №5

# «Шаблоны проектирования и модульное тестирование в Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5Ц-53Б |  | преподаватель каф. ИУ5 |
| Рабинович А.Д. |  | Надрид А.Н. |

Москва, 2024 г.

**Постановка задачи:**

Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.

В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

**Текст программы:**

**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

**TDD.py**

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()

**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}"

**main.py**

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"