Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторная работа №4
по дисциплине «Базовые компоненты интернет технологий»

Выполнил:
студент группы ИУ5-35Б Тазенков И. Д.

г. 2021

Задание

- Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
- Вместо реализации паттерна Вы можете написать тесты для своей программы решения биквадратного уравнения. В этом случае, возможно, Вам потребуется доработать программу решения биквадратного уравнения, чтобы она была пригодна для модульного тестирования.
- В модульных тестах необходимо применить следующие технологии:

  - TDD - фреймворк.
  - BDD - фреймворк.
  - Создание Mock-объектов.

Текст программы

EquationSlover.py

```
# -*- coding: utf-8 -*-

import sys
import math


class EquationSolver:
    def __init__(self):
        self._result = []
        self._ratio = [0, 0, 0]

    @property
    def ratio(self):
        return self._ratio

    @ratio.setter
    def ratio(self, coefs):
        self._ratio = coefs

    @property
    def result(self):
        return self._result

    def input_ratio(self):
        self._addcoef(0, 'Введите коэффициент A:')
        while self._ratio[0] == 0:
            self._addcoef(0, 'Введите коэффициент A:')
        self._addcoef(1, 'Введите коэффициент B:')
        self._addcoef(2, 'Введите коэффициент C:')

    def _addcoef(self, index, prompt):
        try:
            coef_str = sys.argv[index + 1]
            if sys.argv[1] == '0':
                int('bn')
            float(coef_str)
        except:
```

```
            flag = True
            while flag:
                print(prompt)
                coef_str = str(input())
                if coef_str.isdigit() or (coef_str[0] == '-' and
coef_str[1:].isdigit()):
                    flag = False

        self._ratio[index] = float(coef_str)

    def get_roots(self):
        result = set()
        a, b, c = self.ratio
        D = b * b - 4 * a * c
        if D == 0.0:
            root = -b / (2.0 * a)
            if root > 0:
                result.add(math.sqrt(root))
                result.add(-math.sqrt(root))
            elif root == 0:
                result.add(abs(math.sqrt(root)))

        elif D > 0.0:
            root1, root2, root3, root4 = None, None, None, None
            sqD = math.sqrt(D)
            rootSq1 = (-b + sqD) / (2.0 * a)
            rootSq2 = (-b - sqD) / (2.0 * a)

            if rootSq1 > 0:
                root1 = math.sqrt(rootSq1)
                root2 = -math.sqrt(rootSq1)
            elif rootSq1 == 0:
                root1 = abs(math.sqrt(rootSq1))

            if rootSq2 > 0:
                root3 = math.sqrt(rootSq2)
                root4 = -math.sqrt(rootSq2)
            elif rootSq2 == 0:
                root3 = abs(math.sqrt(rootSq2))
            result.add(root1)
            result.add(root2)
            result.add(root3)
            result.add(root4)

        self._result = list(filter(lambda x: x is not None, result))
        return self._result

    def printResult(self):
        if len(self._result) == 0:
            print("Нет корней")
        elif len(self._result) == 1:
            print("Один корень: {}".format(self._result[0]))
        elif len(self._result) == 2:
            print("Два корня: {} и {}".format(self._result[0],
self._result[1]))
        elif len(self._result) == 3:
            print("Три корня: {}, {} и {}".format(self._result[0],
self._result[1], self._result[2]))
        else:
            print("Четыре корня: {}, {}, {} и {}". format(self._result[0],
self._result[1], self._result[2], self._result[3]))


def main():
```

```
    solver = EquationSolver()
    solver.input_ratio()
    solver.get_roots()
    solver.printResult()


if __name__ == "__main__":
    main()
```

TestUnitEquationSolver.py

```
import math
import unittest
from EquationSolver import EquationSolver


class TestEquation(unittest.TestCase):
    def setUp(self):
        self.solver = EquationSolver()

    def test_ratio1(self):
        self.solver.ratio = [1, 1, -20]
        self.assertEqual(self.solver.ratio, [1.0, 1.0, -20.0])

    def test_ratio2(self):
        self.solver.ratio = [0, 0, 0]
        self.assertEqual(self.solver.ratio, [0, 0, 0])

    def test_result(self):
        self.solver.ratio = [1, 1, -20]
        self.solver.get_roots()
        self.assertEqual(sorted(self.solver.result), sorted([-2, 2]))

    def test_result2(self):
        self.solver.ratio = [1, -6, 5]
        self.solver.get_roots()
        self.assertEqual(sorted(self.solver.result), sorted([1, -1,
math.sqrt(5), -math.sqrt(5)]))

    def test_result4(self):
        self.solver.ratio = [1, 1, -1]
        self.solver.get_roots()
        self.assertEqual(sorted(self.solver.result), sorted([-0.5 *
math.sqrt(-2 + 2 * math.sqrt(5)), 0.5 * math.sqrt(-2 + 2 * math.sqrt(5))]))

    def test_result3(self):
        self.solver.ratio = [1, 14, 48]
        self.solver.get_roots()
        self.assertEqual(self.solver.result, [])


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

features/test.feature

```
Feature: Test EquationSolver

  Scenario: Run a simple test
    Given the user enters ratio 1, 1, -20
    When Finding roots
    Then Test roots 2, -2
```

Features/steps/test_equationsolver.py

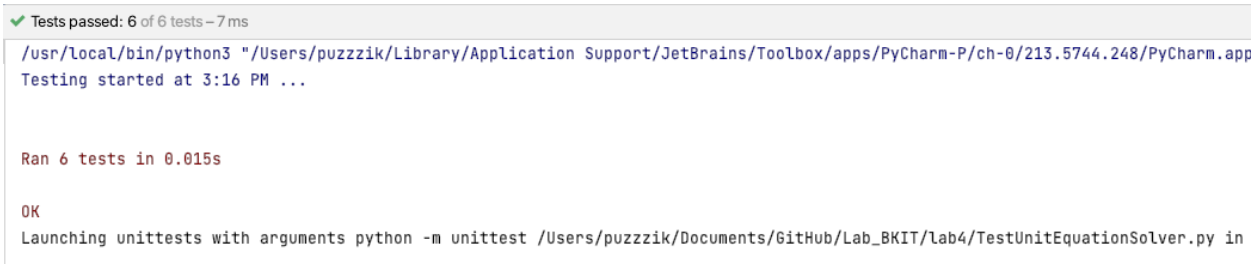
```
from EquationSolver import EquationSolver

from behave import *


@given(u'the user enters ratio {a}, {b}, {c}')
def step_impl(context, a, b, c):
    context.solver = EquationSolver
    context.solver.ratio = list(map(int, [a, b, c]))


@when('Finding roots')
def asd_impl(context):
    context.result = context.solver.get_roots(context.solver)


@then('Test roots {r1}, {r2}')
def abc_impl(context, r1, r2):
    a = sorted(context.result)
    b = sorted(list(map(float, [r1, r2])))

    for i in range(len(a)):
        for j in range(len(b)):
            if i == j:
                assert a[i] == b[j]
```

Mock/test.py

```
from unittest import TestCase
from unittest.mock import patch, Mock
import time
from math import sqrt
from main import EquationSolver


class TestSolver(TestCase):
    @patch('main.EquationSolver.input_ratio', return_value=0)
    def test_solver(self, input_ratio):
        solver = EquationSolver
        solver.ratio = [1, 1, -20]
        print(solver.ratio)
        solver.input_ratio()

        solver.get_roots(self=solver)

        a = solver.getResult(self=solver)
        self.assertEqual(sorted(solver.getResult(self=solver)), [-2, 2])


if __name__ == '__main__':
    TestSolver.test_solver()
```

Результаты

Unittests:



Mock:

```
Launching unittests with arguments python -m unittest /Users/puzzzik/



Ran 1 test in 0.006s

OK
[1, 1, -20]
```

BDD:

```
Feature: Test EquationSolver # test.feature:1

  Scenario: Run a simple test             # test.feature:3
    Given the user enters ratio 1, 1, -20 # steps/test_equationsolver.py:6 0.000s
    When Finding roots                    # steps/test_equationsolver.py:12 0.000s
    Then Test roots 2, -2                 # steps/test_equationsolver.py:17 0.000s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.001s
```