### 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 ИУ5 «Системы обработки информации и управления»**

## Лабораторная работа №4

**«**Шаблоны проектирования и модульное тестирование в Python» по дисциплине «Базовые компоненты интернет-технологий»

## Выполнил: студент группы ИУ5-35Б

## Герасимов А.Д.

2021 г.

# 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.](https://refactoring.guru/ru/design-patterns/catalog) 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.

Лабораторная работа была сделана по пункту 2 и 3.

**Текст программы:**

Main.py

from TDD import TestShapeFunctions  
from shapes import Rectangle, Square  
from shapes import Circle  
from ShapeFactory import ShapeFactory  
from Shape import Shape  
import unittest  
from unittest import TestCase  
  
  
def main():  
  
 shape\_factory = ShapeFactory()  
 shape\_name = input("Enter the name of the shape: ")  
  
 shape = shape\_factory.create\_shape(shape\_name)  
  
 print(f"The type of object created: {type(shape)}")  
 print(f"The area of the {shape\_name} is: {shape.calculate\_area()}")  
 print(f"The perimeter of the {shape\_name} is: {shape.calculate\_perimeter()}")  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

shapes.py

from Shape import Shape  
import abc  
class Rectangle(Shape):  
 def \_\_init\_\_(self, height, width):  
 self.height = height  
 self.width = width  
  
 def calculate\_area(self):  
 return self.height \* self.width  
  
 def calculate\_perimeter(self):  
 return 2 \* (self.height + self.width)  
  
class Square(Shape):  
 def \_\_init\_\_(self, width):  
 self.width = width  
  
 def calculate\_area(self):  
 return self.width \*\* 2  
  
 def calculate\_perimeter(self):  
 return 4 \* self.width  
  
class Circle(Shape):  
 def \_\_init\_\_(self, radius):  
 self.radius = radius  
  
 def calculate\_area(self):  
 return 3 \* self.radius \* self.radius  
  
 def calculate\_perimeter(self):  
 return 2 \* 3.14 \* self.radius

shapesFactory.py

from shapes import Circle  
from shapes import Rectangle  
from shapes import Square  
class ShapeFactory:  
 def create\_shape(self, name):  
 if name == 'circle':  
 radius = input("Enter the radius of the circle: ")  
 return Circle(float(radius))  
  
 elif name == 'rectangle':  
 height = input("Enter the height of the rectangle: ")  
 width = input("Enter the width of the rectangle: ")  
 return Rectangle(int(height), int(width))  
  
 elif name == 'square':  
 width = input("Enter the width of the square: ")  
 return Square(int(width))

shape.py

import abc  
class Shape(metaclass=abc.ABCMeta):  
 @abc.abstractmethod  
 def calculate\_area(self):  
 pass  
  
 @abc.abstractmethod  
 def calculate\_perimeter(self):  
 pass

TDD.py

from shapes import Rectangle, Square  
from shapes import Circle  
import unittest  
  
  
class TestShapeFunctions(unittest.TestCase):  
 def test\_calculate\_area\_of\_rectangle(self):  
 rec = Rectangle(2, 2)  
 self.assertEqual(rec.calculate\_area(), 4, "Should be 4")  
  
 def test\_calculate\_perimeter\_of\_rectangle(self):  
 rec = Rectangle(2, 2)  
 self.assertEqual(rec.calculate\_perimeter(), 8, "Should be 8")  
  
 def test\_calculate\_area\_of\_Square(self):  
 square = Square(2)  
 self.assertEqual(square.calculate\_area(), 4, "Should be 4")  
  
 def test\_calculate\_perimeter\_of\_Square(self):  
 square = Square(2)  
 self.assertEqual(square.calculate\_perimeter(), 8, "Should be 8")  
  
 def test\_calculate\_area\_of\_circle(self):  
 circle = Circle(2)  
 self.assertEqual(circle.calculate\_area(), 12, "Should be 12")  
  
 def test\_calculate\_perimeter\_of\_circle(self):  
 circle = Circle(2)  
 self.assertEqual(circle.calculate\_perimeter(), 12.56, "Should be 12.56")  
  
  
suite = unittest.TestLoader().loadTestsFromTestCase(TestShapeFunctions)  
unittest.TextTestRunner(verbosity=2).run(suite)

steps.py

from behave import given, when, then, step  
from shapes import Rectangle, Square, Circle  
  
  
# rectangle = Rectangle(2, 3)  
  
@given("Rectangle has {height:d} and {width:d} cm")  
def initial\_impl(context, width, height):  
 context.rectangle = Rectangle(width, height)  
  
  
@when("I search area of rectangle")  
def area\_impl(context):  
 context.result = context.rectangle.calculate\_area()  
  
  
@then("I should have area of Rectangle {result:d}")  
def result\_impl(context, result):  
 assert context.result == result  
  
  
# Square  
  
@given("Square has {width:d} cm")  
def initial\_impl(context, width):  
 context.square = Square(width)  
  
  
@when("I search area of square")  
def area\_impl(context):  
 context.result = context.square.calculate\_area()  
  
  
@then("I should have area of square {result:d}")  
def result\_impl(context, result):  
 assert context.result == result  
  
  
# Circle  
@given("Circle has {radius:d} cm")  
def initial\_impl(context, radius):  
 context.circle = Circle(radius)  
  
  
@when("I search area of circle")  
def area\_impl(context):  
 context.result = context.circle.calculate\_area()  
  
  
# Не сравнивает дробные значения!  
@then("I have area of circle {result:d}")  
def result\_impl(context, result):  
 assert context.result == result

square.feature

Feature: Square  
 Scenario: aboba  
 Given Square has **4** cm  
 When I search area of square  
 Then I should have area of square **16**

circle.feature

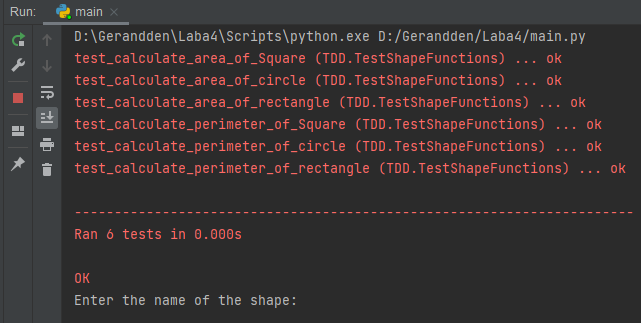
Feature: Circle  
 Scenario: Arguments for given, when, then  
 Given circle has **2** cm  
 When I search area of circle  
 Then I have area of circle **12**

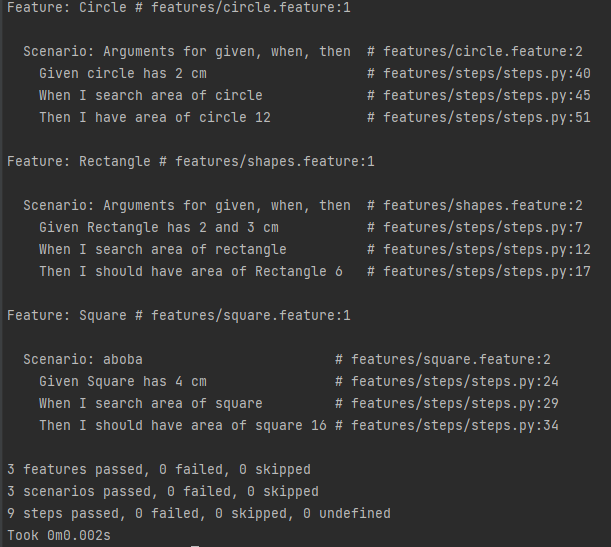
rectangle.feature

Feature: Rectangle  
 Scenario: Arguments for given, when, then  
 Given Rectangle has **2** and **3** cm  
 When I search area of rectangle  
 Then I should have area of Rectangle **6**

**Результаты тестирования:**

TDD:



BDD: 

**Результат выполнения программы:**

