**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

по дисциплине «Базовые компоненты интернет-технологий»

Выполнила:

студентка группы ИУ5-33Б

Жамнова М. С.

Проверил:

Канев А.И.

2021 г.

### Задание:

1. 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.
2. 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.
3. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Текст программы**

composite.py

from abc import ABC, abstractmethod  
  
  
class IterPart(ABC):  
  
 # Интерфейс частей компьютера  
  
 @abstractmethod  
 def name(self) -> str:  
 pass  
  
 @abstractmethod  
 def cost(self) -> float:  
 pass  
  
  
class Part(IterPart):  
  
 # Класс части компьютера  
  
 def \_\_init\_\_(self, name: str, cost: float):  
 try:  
 self.\_\_cost = float(cost)  
 except:  
 self.\_\_cost = 0  
 self.\_\_name = name  
  
 def cost(self) -> float:  
 return self.\_\_cost  
  
 def name(self) -> str:  
 return self.\_\_name  
  
  
class ComplexPart(IterPart):  
  
 # Класс составных частей ноутбука  
  
 def \_\_init\_\_(self, name: str):  
 self.\_\_name = name  
 self.parts = []  
  
 def cost(self):  
 cost = 0  
 for i in self.parts:  
 cost += i.cost()  
 return cost  
  
 def name(self) -> str:  
 return self.\_\_name  
  
 def add\_product(self, part: IterPart):  
 self.parts.append(part)  
  
 def remove\_product(self, part: IterPart):  
 self.parts.remove(part)  
  
 def clear(self):  
 self.parts = []  
  
  
class Comp (ComplexPart):  
  
 # Класс компьютера, состоящий из частей  
  
 def \_\_init\_\_(self, name: str):  
 super(Comp, self).\_\_init\_\_(name)  
  
 def cost(self):  
 cost = 0  
 for i in self.parts:  
 cost\_i = i.cost()  
 print(f"Price for '{i.name()}' is {cost\_i} рублей")  
 cost += cost\_i  
  
 print(f"Price for '{self.name()}' is {cost} рублей")  
 return cost  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 print('Pattern Composite \n')  
 ControlDivices = ComplexPart('Control divices')  
 ControlDivices.add\_product(Part('Mouse', 800))  
 ControlDivices.add\_product(Part('Keyboard', 1920))  
 Monitor = Part('Monitor HP', 50700)  
 SystemUnit = Part('System unit HP', 23400)  
 Equipment = Comp('HP')  
 Equipment.add\_product(ControlDivices)  
 Equipment.add\_product(Monitor)  
 Equipment.add\_product(SystemUnit)  
 print(Equipment.cost())

TDD\_test.py

import unittest  
import sys, os  
from composite import \*  
  
sys.path.append(os.getcwd())  
  
class TestPartCost(unittest.TestCase):  
 def test\_part\_cost\_is\_working(self):  
 ControlDivices = ComplexPart('Control divices')  
 ControlDivices.add\_product(Part('Mouse', 800))  
 ControlDivices.add\_product(Part('Keyboard', 1920))  
 self.assertEqual(ControlDivices.cost(), 2720)  
  
 def test\_part\_cost\_receives\_string\_is\_working(self):  
 ControlDivices = ComplexPart('Control divices')  
 ControlDivices.add\_product(Part('Mouse', '800'))  
 ControlDivices.add\_product(Part('Keyboard', '1920'))  
 self.assertIsInstance(ControlDivices.cost(), float)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

myfeatures.steps.py

from behave import \*  
from tdd\_test.TDD\_test import \*  
  
@given("I have Mouse for 800 rubles and Keyboard for 1920 rubles")  
def have\_prices(context):  
 context.a = TestPartCost()  
  
  
@when("I put them into ControlDivices")  
def ControlDivices\_combine(context):  
 context.a.test\_part\_cost\_is\_working()  
  
  
@then("I expect ControlDivices to cost 2720 rubles")  
def check\_result(context):  
 pass

myfeature.feature

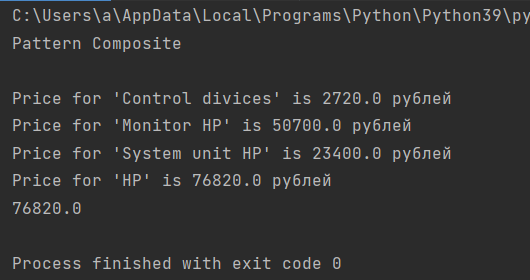
Feature: Test  
 Scenario: Test function  
 Given I have Mouse for 800 rubles and Keyboard for 1920 rubles  
 When I put them into ControlDivices  
 Then I expect ControlDivices to cost 2720 rubles

moke\_test.py

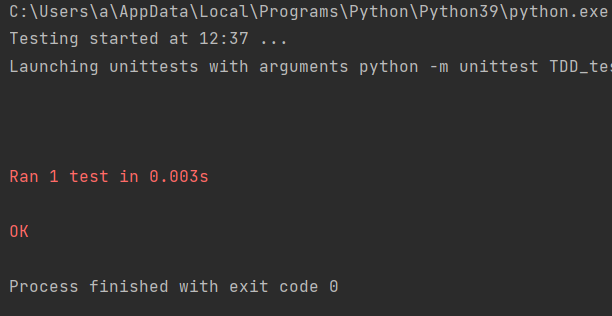
import unittest  
import sys, os  
from unittest.mock import patch, Mock  
  
import composite  
  
sys.path.append(os.getcwd())  
from composite import \*  
  
  
class TestComposite(unittest.TestCase):  
 @patch.object(composite.Comp, 'cost')  
 def test\_comp\_cost(self, mock\_cost):  
 mock\_cost.return\_value = "100"  
 Equipment = Comp ('HP')  
 self.assertEqual(Equipment.cost(), "100")

**Скрины результатов**

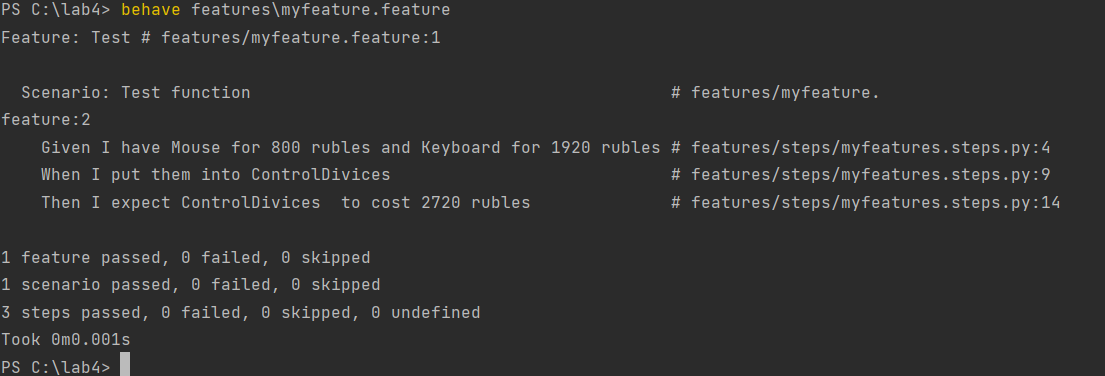
Результат работы composite.py



Результат TDD – фреймфорк



Результат BDD – фреймфорк



Результат moke\_test.py

