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 «ГУИМЦ»
КАФЕДРА ИУ5 «Системы обработки информации и управления»

Дисциплина «Базовые компоненты интернет-технологий»
ОТЧЕТ

Лабораторная работа №4
«Шаблоны проектирования и модульное тестирование в Python»

Студент: Соловьева А.М., группа ИУ5Ц-53Б
Преподаватель: Гапанюк Ю.Е.

2021 г.

СОДЕРЖАНИЕ

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

1. Описание задания:

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. 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.

2. Листинг программы и результаты:

bulder.py

```
from abc import ABC, abstractmethod
from enum import Enum, auto


class BrandType(Enum):
    Apple = auto()
    Xiaomi = auto()
    Samsung = auto()
    Huawei = auto()

class MemoryType(Enum):
    M64 = auto()
    M128 = auto()
    M256 = auto()

class MatrType(Enum):
    IPS = auto()
    Amoled = auto()

class DiagType(Enum):
    D54 = auto()
    D58 = auto()
    D61 = auto()
    D65 = auto()

class Smartphone:
    def __init__(self, name):
        self.name = name
        self.brand = None
        self.memory = None
```

```
        self.matr = None
        self.diag = None
        self.cost = None

    def __str__(self):
       info: str = f"Smartphone name: {self.name} \n" \
                   f"{self.brand} \n" \
                   f"{self.memory} \n" \
                   f"{self.matr} \n" \
                   f"{self.diag} \n" \
                   f"Cost: {self.cost} rub"

       return info



class Builder(ABC):

    @abstractmethod
    def add_brand(self) -> None: pass

    @abstractmethod
    def add_memory(self) -> None: pass

    @abstractmethod
    def add_matr(self) -> None: pass

    @abstractmethod
    def add_diag(self) -> None: pass


class Apple11Builder(Builder):

    def __init__(self):
        self.smartphone = Smartphone("Apple 11")
        self.smartphone.cost = 60000

    def add_brand(self) -> None:
        self.smartphone.brand = BrandType.Apple

    def add_memory(self) -> None:
        self.smartphone.memory = MemoryType.M128

    def add_matr(self) -> None:
        self.smartphone.matr = MatrType.IPS

    def add_diag(self) -> None:
        self.smartphone.diag = DiagType.D61

    def get_lap(self) -> Smartphone:
        return self.smartphone


class Huawei50LiteBuilder(Builder):

    def __init__(self):
        self.smartphone = Smartphone("Huawei 50 Lite")
        self.smartphone.cost = 25000

    def add_brand(self) -> None:
        self.smartphone.brand = BrandType.Huawei

    def add_memory(self) -> None:
```

```
        self.smartphone.memory = MemoryType.M256

    def add_matr(self) -> None:
        self.smartphone.matr = MatrType.Amoled

    def add_diag(self) -> None:
        self.smartphone.diag = DiagType.D65

    def get_lap(self) -> Smartphone:
        return self.smartphone


class Director:
    def __init__(self):
        self.builder = None

    def set_builder(self, builder: Builder):
        self.builder = builder

    def make_lap(self):
        if not self.builder:
            raise ValueError("Builder didn't set")
        self.builder.add_brand()
        self.builder.add_memory()
        self.builder.add_matr()
        self.builder.add_diag()


def check_cost(name1):
    for it1 in (Apple11Builder, Huawei50LiteBuilder):
        director1 = Director()
        builder1 = it1()
        director1.set_builder(builder1)
        director1.make_lap()
        smartphone1 = builder1.get_lap()
        if smartphone1.name == name1:
            return smartphone1.cost

def sum_cost(x):
    for it1 in (Apple11Builder, Huawei50LiteBuilder):
        director1 = Director()
        builder1 = it1()
        director1.set_builder(builder1)
        director1.make_lap()
        smartphone1 = builder1.get_lap()
        x = x + smartphone1.cost
    return x


if __name__ == "__main__":
    print("Объекты:")
    director = Director()
    for it in (Apple11Builder, Huawei50LiteBuilder):
        builder = it()
        director.set_builder(builder)
        director.make_lap()
        smartphone = builder.get_lap()
        print(smartphone)
        print('---------------')
    name = "Huawei 50 Lite"
    print(name, "Cost:", check_cost(name))
    x = 0
    print('sum = ', sum_cost(x))
```

```
C:\Users\sashu\PycharmProjects\pyth
Объекты:
Smartphone name: Apple 11
BrandType.Apple
MemoryType.M128
MatrType.IPS
DiagType.D61
Cost: 60000 rub
---------------
Smartphone name: Huawei 50 Lite
BrandType.Huawei
MemoryType.M256
MatrType.Amoled
DiagType.D65
Cost: 25000 rub
---------------
Huawei 50 Lite Cost: 25000
sum =  85000
```

tdd.py

```python
import unittest
import sys, os

sys.path.append(os.getcwd())
from builder import *

class TestCost(unittest.TestCase):

    def test_cost(self):
        self.assertEqual(check_cost("Huawei 50 Lite"), 25000)

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

```
Ran 1 test in 0.002s

OK
Launching unittests with arguments python -m unittest tdd.TestCost in C:\Users\sashu\PycharmProjects\python_LAB№4
```

build.feature

```
Feature: Test

  Scenario: Test sum_cost
    Given I have sum = 0
    When I sum the cost
    Then I expect to get result = 60000
```

steps.py

```python
from behave import given, when, then
from builder import *

@given('I have sum = {x:g}')
def step(context, x):
    context.x = x

@when('I sum the cost')
def step(context):
    context.x = sum_cost(context.x)

@then('I expect to get result = {result:g}')
def step(context, result):
    assert context.x == result
```

```
C:\Users\sashu\PycharmProjects\python_LAB№4\venv\Scripts\python

Process finished with exit code 0
```

mock.py

```python
from builder import *
from unittest import TestCase
from unittest.mock import patch

class TestCost(TestCase):
    @patch('builder.sum_cost', return_value=85000)
    def test_sum_cost(self, x):
        self.assertEqual(sum_cost(0), 85000)
```

```
Testing started at 1:34 ...
Launching unittests with arguments python -m unittest mock.TestCost.test_sum_cost in
 C:\Users\sashu\PycharmProjects\python_LAB№4



Ran 1 test in 0.002s

OK

Process finished with exit code 0
```