Московский государственный технический университет им. Н.Э. Баумана

Факультет “Радиотехнический”
Кафедра ИУ5 “Системы обработки информации и управления”

Отчет по лаб 4
“Шаблоны проектирования и модульное тестирование в Python.”
Базовые компоненты интернет технологий

Вариант 8

Подготовил:
Студент группы РТ5-31Б
Коваленко В.И.

Проверил:
Доцент кафедры ИУ5
Гапанюк Ю.Е.

12 Декабря 2021г.

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

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.

Листинг

Builder.py

```
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Any


class Builder(ABC):

    @property
    @abstractmethod
    def car(self) -> None:
        pass

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

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

    @abstractmethod
    def turbo(self) -> None:
        pass
```

```
    @abstractmethod
    def body(self) -> None:
        pass


class CarBuilder(Builder):
    def __init__(self) -> None:
        self.reset()

    def reset(self) -> None:
        self._product = Car()

    @property
    def car(self) -> Car:
        product = self._product
        self.reset()
        return product

    def wheels(self) -> None:
        self._product.add("Колеса")

    def engine(self) -> None:
        self._product.add("Двигатель")

    def body(self) -> None:
        self._product.add("Кузов")
    def turbo(self) -> None:
        self._product.add("Турбина")
class Car():
    def __init__(self) -> None:
        self.parts = []

    def add(self, part: Any) -> None:
        self.parts.append(part)

    def list_parts(self) -> None:
        print(f"Части автомобиля: {', '.join(self.parts)}", end="")


class Director:
    def __init__(self) -> None:
        self._builder = None

    @property
```

```
    def builder(self) -> Builder:
        return self._builder
    @builder.setter
    def builder(self, builder: Builder) -> None:
        self._builder = builder

    def TurboCar(self) -> None:
        self.builder.wheels()
        self.builder.engine()
        self.builder.body()
        self.builder.turbo()

    def AtmoCar(self) -> None:
        self.builder.wheels()
        self.builder.engine()
        self.builder.body()

if __name__ == "__main__":
    director = Director()
    builder = CarBuilder()
    director.builder = builder

    print("Турбовый автомобиль: ")
    director.TurboCar()
    builder.car.list_parts()
    print("\n")
    print("Автомобиль без турбины: ")
    director.AtmoCar()
    builder.car.list_parts()
```

```
Турбовый автомобиль:
Части автомобиля: Колеса, Двигатель, Кузов, Турбина

Автомобиль без турбины:
Части автомобиля: Колеса, Двигатель, Кузов%
kovalenkov@MacBook-Air-Vlad Лаб 4 %
```

test_mock.py

```
import unittest
import sys, os
from unittest.main import main
from unittest.mock import patch, Mock

import Builder
sys.path.append(os.getcwd())
from Builder import *
```

```
class Test_builder(unittest.TestCase):
    @patch.object(builder.Director, 'TurboCar')
    def test_turbocar(self, mock_turbocar):
        mock_turbocar.return_value = None

        director = Director()
        builder = CarBuilder()
        director.builder = builder

        self.assertEqual(director.TurboCar(), None)
if __name__ == "__main__":
    unittest.main()
```

test_TDD.py

```
import unittest
from Builder import *

class TestBuilder(unittest.TestCase):
    def Test_builder(self):
        builder = ()
        self.assertEqual(builder.create(), None)

    def test_wheels(self):
        builder = CarBuilder()
        self.assertEqual(builder.wheels(), None)


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