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»

Выполнил:
студент группы ИУ5-33Б
Некрасов С. А.

Проверил:
Канев А.И.

2021 г.

Оглавление

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. 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:

Файл builder.py:

```
from __future__ import annotations  # для аннотаций ->
from abc import ABC, abstractmethod  # для использования абстракции
from typing import Any  # Средство проверки статического типа


# для значения типа Any и присвоить его любой переменной


# Строитель — это порождающий паттерн проектирования, который позволяет
создавать сложные объекты пошагово.
# Строитель даёт возможность использовать один и тот же код строительства для
получения разных представлений объектов.
class Builder(ABC):

    @property  # property позволяет превратить метод класса в атрибут класса

    @abstractmethod  # Абстрактным называется объявленный, но не
реализованный метод
    def product(self) -> None:  # проверка типов аргументов и возвращаемое
значение функции
        pass

    @abstractmethod  # Абстрактным называется объявленный, но не
реализованный метод
    def milk(self) -> None:  # молоко
        pass

    @abstractmethod  # Абстрактным называется объявленный, но не
реализованный метод
    def gum(self) -> None:  # жвачка
        pass

    @abstractmethod  # Абстрактным называется объявленный, но не
реализованный метод
    def washing_powder(self) -> None:  # стиральный порошок
```

```
        pass


class Shop_Builder(Builder):  # мой класс - строитель магазина

    def __init__(self) -> None:
        self.reset()

    def reset(self) -> None:  # функция сброса
        self._product = Shop()

    @property  # property позволяет превратить метод класса в атрибут класса
    def product(self) -> Shop:
        product = self._product
        self.reset()
        return product

    def milk(self) -> None:
        self._product.add("молоко")

    def gum(self) -> None:
        self._product.add("жвачка")

    def washing_powder(self) -> None:
        self._product.add("стиральный порошок")


class Shop:

    def __init__(self) -> None:
        self.parts = []  # parts - части

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

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


class Director:  # Директор отвечает только за выполнение шагов построения в
определённой последовательности

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

    @property  # property позволяет превратить метод класса в атрибут класса
    def builder(self) -> Builder:
        return self._builder

    @builder.setter  # применяется сеттер к методу builder, то есть делаем
метод доступным для записи
    def builder(self, builder: Builder) -> None:
        self._builder = builder

    def Magnit(self) -> None:
        self.builder.milk()
        self.builder.gum()

    def Magnit_Cosmetic(self) -> None:
        self.builder.gum()
        self.builder.washing_powder()


if __name__ == "__main__":
```

```
    director = Director()  # наследование класса
    builder = Shop_Builder()
    director.builder = builder

    print("Магнит: ")
    director.Magnit()
    builder.product.list_parts()

    print('\n')

    print("Магнит косметик: ")
    director.Magnit_Cosmetic()
    builder.product.list_parts()
```

Файл test_TDD:

```
import unittest  # автоматизация тестов
import sys, os  # предоставляет системе особые параметры и функции

from builder import *  # импортировать всё из builder

sys.path.append(os.getcwd())  # добавить путь поиска модулей


class Shop_Test_Builder(unittest.TestCase):
    builder = Shop_Builder()

    def test_magnit_builder(self):
        self.assertEqual(Director.Magnit(self), None)

    def test_magnit_cosmetic_builder(self):
        self.assertEqual(Director.Magnit_Cosmetic(self), None)
```

Файл my_feature.feature:

```
Feature: Test
  Scenario: Test_Builder
    Given Shop_Builder
    When test_magnit_builder return OK
    And test_magnit_cosmetic_builder return OK
    Then Successfully
```

Файл test_BDD:

```
from behave import *  # импортировать всё из behave

from test_TDD import *  # импортировать всё из test_TDD


@given('Shop_Builder')
def first_step(context):
    context.a = Shop_Test_Builder()


@when('test_magnit_builder return OK')
def test_magnit_builder(context):
    context.a.test_magnit_builder()
```

```
@when('test_magnit_cosmetic_builder return OK')
def test_magnit_cosmetic_builder(context):
    context.a.test_magnit_cosmetic_builder()


@then('Successfully')
def last_step(context):
    pass
```

Файл test_Mock:

```
import unittest  # автоматизация тестов
import sys, os  # предоставляет системе особые параметры и функции
from unittest.mock import patch, Mock
# Когда функция оформлена через @patch, mock класса, метода или функции,
# переданная в качестве цели для @patch, возвращается и передается в качестве
аргумента декорируемой функции.
import builder

sys.path.append(os.getcwd())  # добавить путь поиска модулей
from builder import *


class Shop_Test_Builder(unittest.TestCase):
    @patch.object(builder.Shop_Builder(), 'milk')
    # patch.object принимает объект и имя атрибут, который требуется
исправить,
    # а также, при необходимости, значение для исправления.
    def test_milk(self, mock_milk):
        mock_milk.return_value = None
        self.assertEqual(Shop_Builder().milk(), None)
```

Экранные формы с примерами выполнения программы:

Реализация порождающего шаблона builder:

```
Магнит:
В магазине продаются: молоко, жвачка


Магнит косметик:
В магазине продаются: жвачка, стиральный порошок
Process finished with exit code 0
```

Тестирование (TDD – фреймворк):

```
C:\Users\79508\PycharmProjects\Lab4\venv\Scripts\python.exe
Testing started at 16:50 ...


Ran 2 tests in 0.002s

OK
Launching unittests with arguments python -m unittest test_T


Process finished with exit code 0
```

Тестирование (BDD – фреймворк):

```
PS C:\Users\79508\PycharmProjects\Lab4> behave Features\my_feature.feature
Feature: Test # Features/my_feature.feature:1

  Scenario: Test_Builder                          # Features/my_feature.feature:2
    Given Shop_Builder                            # Features/steps/test_BDD.py:6
    When test_magnit_builder return OK            # Features/steps/test_BDD.py:11
    And test_magnit_cosmetic_builder return OK # Features/steps/test_BDD.py:16
    Then Successfully                             # Features/steps/test_BDD.py:21

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
4 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
```

Тестирование (Создание Mock-объектов):

```
C:\Users\79508\PycharmProjects\Lab4\venv\Scripts\pytho
Testing started at 16:47 ...
Launching unittests with arguments python -m unittest



Ran 1 test in 0.002s

OK

Process finished with exit code 0
```