**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф.ИУ5 |
| Кондрахин Сергей Сергеевич |  | Гапанюк Юрий  Евгеньевич |
|  |  |  |

Москва, 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. Для сдачи лабораторной работы в минимальной варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:

* TDD – фреймворк
* BDD – фреймворк
* Создание Mock-объектов

**Текст программы**

**builder.py**

from \_\_future\_\_ import annotations   
from abc import ABC, abstractmethod   
from typing import Any   
  
class Builder(ABC):  
 @property   
  
 @abstractmethod   
 def product(self) -> None:   
 pass  
  
 @abstractmethod  
 def buns(self) -> None: # булочки  
 pass  
  
 @abstractmethod  
 def cutlet(self) -> None: # котлета  
 pass  
  
 @abstractmethod  
 def pickles(self) -> None: # огурчики  
 pass  
  
 @abstractmethod  
 def cheese(self) -> None: # сыр  
 pass  
  
 @abstractmethod  
 def ketchup(self) -> None: # кетчуп  
 pass  
  
class Burger\_Builder(Builder): # строитель бургера  
 def \_\_init\_\_(self) -> None:  
 self.reset()  
  
 def reset(self) -> None:   
 self.\_product = Burger()  
  
 @property   
 def product(self) -> Burger:  
 product = self.\_product  
 self.reset()  
 return product  
  
 def buns(self) -> None:  
 self.\_product.add("булочки")  
  
 def cutlet(self) -> None:  
 self.\_product.add("котлета")  
  
 def pickles(self) -> None:  
 self.\_product.add("огурчики")  
  
 def cheese(self) -> None:  
 self.\_product.add("сыр")  
  
 def ketchup(self) -> None:  
 self.\_product.add("кетчуп")  
  
class Burger:  
 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 состав входят: {', '.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 Hamburger(self) -> None:  
 self.builder.buns()  
 self.builder.cutlet()  
 self.builder.pickles()  
 self.builder.ketchup()  
  
 def Cheeseburger(self) -> None:  
 self.builder.buns()  
 self.builder.cutlet()  
 self.builder.cheese()  
 self.builder.ketchup()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 director = Director()   
 builder = Burger\_Builder()  
 director.builder = builder  
 print("Гамбургер:")  
 director.Hamburger()  
 builder.product.list\_parts()  
 print('\n')  
 print("Чизбургер:")  
 director.Cheeseburger()  
 builder.product.list\_parts()

**test\_TDD.py**

import unittest   
import sys, os   
  
from builder import \*   
  
sys.path.append(os.getcwd())   
  
class Burger\_Test\_Builder(unittest.TestCase):  
 builder = Burger\_Builder()  
  
 def test\_hamburger\_builder(self):  
 self.assertEqual(Director.Hamburger(self), None)  
  
 def test\_cheeseburger\_builder(self):  
 self.assertEqual(Director.Cheeseburger(self), None)

**builder.feature**

Feature: Test  
 Scenario: Test\_Builder  
 Given Burger\_Builder  
 When test\_hamburger\_builder return OK  
 And test\_cheeseburger\_builder return OK  
 Then Successfully

**test\_BDD.py**

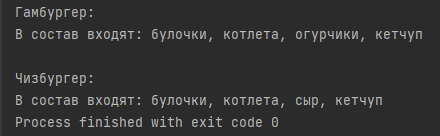
from behave import \*   
from test\_TDD import \*   
  
@given('Burger\_Builder')  
def first\_step(context):  
 context.a = Burger\_Test\_Builder()  
  
@when('test\_hamburger\_builder return OK')  
def test\_hamburger\_builder(context):  
 context.a.test\_hamburger\_builder()  
  
@when('test\_cheeseburger\_builder return OK')  
def test\_cheeseburger\_builder(context):  
 context.a.test\_cheeseburger\_builder()  
  
@then('Successfully')  
def last\_step(context):  
 pass

**test\_Mock.py**

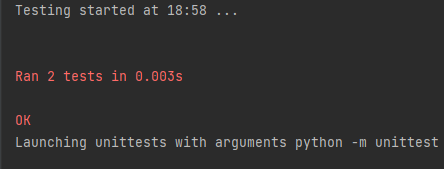
import unittest   
import sys, os   
from unittest.mock import patch, Mock  
import builder  
  
sys.path.append(os.getcwd())   
from builder import \*  
  
class Burger\_Test\_Builder(unittest.TestCase):  
 @patch.object(builder.Burger\_Builder(), 'buns')  
 def test\_buns(self, mock\_buns):  
 mock\_buns.return\_value = None  
 self.assertEqual(Burger\_Builder().buns(), None)

**Примеры выполнения программы**

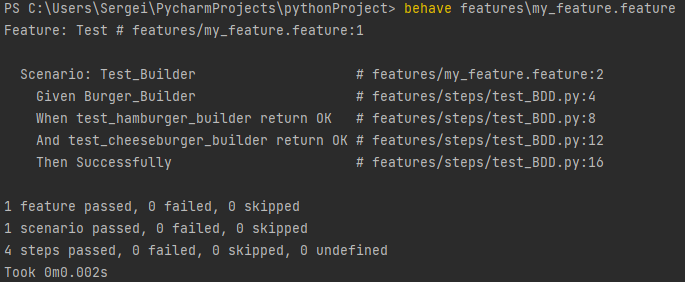
**builder.py**



**test\_TDD.py**



**test\_BDD.py**



**test\_Mock.py**

