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 №3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:

студент группы ИУ5-35Б

Емельянова Т.И.

2021

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.

main.py

```
from burger import *


def work():
    director = Director()
    builder = JustBurger()
    director.builder = builder
    director.create_burger()
    builder.burger.get_description()


def main():
    work()


if __name__ == '__main__':
    main()
```

burger.py

```
from abc import ABC, abstractmethod
from additives import *
from buns import TopBun, BottomBun


class Burger:
    def __init__(self):
        self._kind = bread_kinds.find("Белый хлеб")
```

```
        self._name = ''
        self._parts = []

    @property
    def kind(self):
        return self._kind

    @kind.setter
    def kind(self, kind):
        self._kind = kind

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, name):
        self._name = name

    def add(self, part):
        self._parts.append(part)

    def get_description(self, telegram=0):
        telegram_text = []
        if telegram == 0:
            print(self._name)
        for part in self._parts:
            if telegram == 0:
                print(part.name, part.get_price())
            telegram_text.append(part.name + ', ' + str(part.get_price()))
        if telegram == 0:
            print('Итого:', self.get_price())
        telegram_text.append('Итого: ' + str(self.get_price()))
        return telegram_text

    def get_price(self):
        price = 0
        for part in self._parts:
            price += part.get_price()
        return price


class Builder(ABC):
    @property
    @abstractmethod
    def burger(self, **params):
        pass

    @abstractmethod
```

```
    def get_name(self):
        pass

    @abstractmethod
    def create_base(self, **params):
        pass

    @abstractmethod
    def create_ingredients(self, **params):
        pass

    @abstractmethod
    def create_bottom(self, **params):
        pass


class JustBurger(Builder):
    def __init__(self):
        self._burger = Burger()

    def reset(self):
        self._burger = Burger()

    @property
    def burger(self):
        burger = self._burger
        self.reset()
        return burger

    def get_name(self):
        self._burger.name = 'Простой бургер'

    def create_base(self):
        self._burger.add(TopBun(self._burger.kind, seasoning.find("Кунжут")))

    def create_ingredients(self):
        self._burger.add(base_additives.find("Салат"))
        self._burger.add(base_additives.find("Помидор"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Огурцы"))
        self._burger.add(sauces.find("Кетчуп"))
        self._burger.add(base_additives.find("Сыр чеддер"))

    def create_bottom(self):
        self._burger.add(cutlets.find("Куриная котлета"))
        self._burger.add(BottomBun(self._burger.kind))


class CheeseBurger(JustBurger):
```

```
    def get_name(self):
        self._burger.name = 'Чизбургер'

    def create_base(self):
        self._burger.add(TopBun(self._burger.kind, seasoning.find("Кунжут")))

    def create_ingredients(self):
        self._burger.add(base_additives.find("Салат"))
        self._burger.add(base_additives.find("Помидор"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Огурцы"))
        self._burger.add(sauces.find("Кетчуп"))
        self._burger.add(sauces.find("Горчица"))
        self._burger.add(base_additives.find("Сыр чеддер"))

    def create_bottom(self):
        self._burger.add(cutlets.find("Котлета из говядины"))
        self._burger.add(BottomBun(self._burger.kind))


class MiniCheeseBurger(CheeseBurger):
    def get_name(self):
        self._burger.name = 'Маленький чизбургер'

    def create_base(self):
        self._burger.add(TopBun(self._burger.kind))

    def create_ingredients(self):
        self._burger.add(base_additives.find("Огурцы"))
        self._burger.add(sauces.find("Кетчуп"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Сыр чеддер"))


class DoubleCheeseBurger(CheeseBurger):
    def get_name(self):
        self._burger.name = 'Двойной чизбургер'

    def create_ingredients(self):
        self._burger.add(base_additives.find("Салат"))
        self._burger.add(base_additives.find("Помидор"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Огурцы"))
        self._burger.add(sauces.find("Кетчуп"))
        self._burger.add(sauces.find("Горчица"))
        self._burger.add(base_additives.find("Сыр чеддер"))
        self._burger.add(cutlets.find("Котлета из говядины"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Сыр чеддер"))
```

```
class VeryCheeseBurger(JustBurger):
    def get_name(self):
        self._burger.name = 'Сырный бургер'

    def create_ingredients(self):
        self._burger.add(base_additives.find("Салат"))
        self._burger.add(base_additives.find("Помидор"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Огурцы"))
        self._burger.add(sauces.find("Кетчуп"))
        self._burger.add(sauces.find("Горчица"))
        self._burger.add(base_additives.find("Сыр чеддер"))
        self._burger.add(cutlets.find("Котлета из говядины"))
        self._burger.add(base_additives.find("Лук"))
        self._burger.add(base_additives.find("Сыр чеддер"))


class UserBuilder(Builder):
    def __init__(self, parts):
        self._burger = Burger()
        self.parts = parts

    def reset(self):
        self._burger = Burger()

    @property
    def burger(self):
        burger = self._burger
        self.reset()
        return burger

    def get_name(self):
        self._burger.name = 'Ваш бургер'

    def create_base(self):
        self._burger.add(TopBun(self._burger.kind))

    def create_ingredients(self):
        for part in self.parts:
            base_add = base_additives.find(part)
            sauce = sauces.find(part)
            if base_add.get_price() != 0:
                self._burger.add(base_additives.find(part))
            if sauce.get_price() != 0:
                self._burger.add(sauces.find(part))
        self._burger.add(cutlets.find('Куриная котлета'))
```

```
    def create_bottom(self):
        self._burger.add(BottomBun(self._burger.kind))


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

    @property
    def builder(self):
        return self._builder

    @builder.setter
    def builder(self, builder):
        self._builder = builder

    def create_burger(self):
        self._builder.get_name()
        self._builder.create_base()
        self._builder.create_ingredients()
        self._builder.create_bottom()
```

ingredient.py

```
import sys
from abc import ABC, abstractmethod


class Ingredient(ABC):

    @abstractmethod
    def get_price(self):
        pass
```

burger additive.py

```
from ingredient import Ingredient


class BurgerAdditive(Ingredient):
    def __init__(self, name, price):
        self._name = name
        self._price = price

    def get_price(self):
        return self._price

    @property
```

```
    def name(self):
        return self._name
```

buns.py

```
from ingredient import Ingredient


class Bun(Ingredient):
    def __init__(self, weight, kind, name):
        self._weight = weight
        self._kind = kind
        self._name = name

    def get_price(self):
        return self._weight * self._kind.get_price()

    @property
    def name(self):
        return self._name


class TopBun(Bun):
    def __init__(self, kind, additive=None):
        name = 'Верхняя булочка, ' + kind.name
        if additive is not None:
            name += ', ' + additive.name
        super().__init__(1, kind, name)
        self._additive = additive

    def get_price(self):
        price = super().get_price()
        if self._additive is not None:
            price += self._additive.get_price()
        return price


class BottomBun(Bun):
    def __init__(self, kind):
        super().__init__(0.5, kind, 'Нижняя булочка, ' + kind.name)

    def get_price(self):
        return super().get_price()
```

additives.py

```
from burger_additive import BurgerAdditive
from abc import ABC, abstractmethod
import itertools
```

```
class Components:
    def __init__(self, file_name):
        self.All = []
        self.file_name = file_name
        with open(self.file_name, "r", encoding="utf-8") as file:
            for name, cost in itertools.zip_longest(*[file]*2):
                self.All.append(BurgerAdditive(name.rstrip(), int(cost)))

    def print_all(self, telegram=0):
        telegram_text = []
        index = 1
        for element in self.All:
            text = "{}. {}, {}".format(index, element.name, element.get_price())
            if telegram == 0:
                print(text)
            telegram_text.append(text)
            index += 1
        return telegram_text

    def find(self, name):
        for element in self.All:
            if element.name == name:
                return element
        return BurgerAdditive("", 0)


base_additives = Components("additives/base_additives.txt")
sauces = Components("additives/sauces.txt")
bread_kinds = Components("additives/bread_kinds.txt")
cutlets = Components("additives/cutlets.txt")
seasoning = Components("additives/seasoning.txt")
```

additives/base_additives.txt

```
Помидор
5
Огурцы
5
Лук
3
Салат
7
Сыр чеддер
5
Сыр пармезан
6
Бекон
7
```

```
Халапеньо
5
```

additives/bread_kinds.txt

```
Белый хлеб
10
Чёрный хлеб
20
```

additives/cutlets.txt

```
Куриная котлета
20
Котлета из говядины
30
```

additives/sauces.txt

```
Сырный соус
5
Кетчуп
5
Горчица
5
```

additives/seasoning.txt

```
Кунжут
5
Перец
5
```

TDD_testing.py

```python
import unittest
from additives import Components


class TextComponents(unittest.TestCase):
    def setUp(self):
        self.base_additives = Components("additives/base_additives.txt")

    def test_price(self):
        self.assertEqual(self.base_additives.find("Помидор").get_price(), 5)

    def test_find_null(self):
        self.assertEqual(self.base_additives.find("").get_price(), 0)


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

Mock_testing.py

```python
import unittest
from unittest.mock import patch


class TextComponents(unittest.TestCase):
    @patch('additives.sauces.find', return_value=5)
    def test_price(self, find):
        self.assertEqual(find("Кунжут"), 5)


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

features/steps/BDD_testing_case.py

```python
from behave import *
from additives import seasoning


@given('1')
def step_impl(context):
    context.name = 'Кунжут'
    pass


@when('right')
def step_impl(context):
    context.roots = seasoning.find(context.name).get_price()
    pass


@then('roots')
def step_impl(context):
    assert context.roots == 5
    pass
```

features/get_price.feature

```
Feature: get_price function

 Scenario: test_right_price
    Given 1
    When right
    Then roots
```

Результаты выполнения main.py:

```
Простой бургер
Верхняя булочка, Белый хлеб, Кунжут 15
Салат 7
Помидор 5
Лук 3
Огурцы 5
Кетчуп 5
Сыр чеддер 5
Куриная котлета 20
Нижняя булочка, Белый хлеб 5.0
Итого: 70.0

Process finished with exit code 0
```

Результаты выполнения TDD_testing.py:

```
Ran 2 tests in 0.014s

Launching unittests with arguments py
OK


Process finished with exit code 0
```

Результаты выполнения Mock testing.py:

```
Ran 1 test in 0.009s

OK

Process finished with exit code 0
```

BDD-тестирование:

```
(venv) C:\Users\Lenovo\PycharmProjects\Lab4_Patterns>behave
Feature: get_price function # features/get_price.feature:1

  Scenario: test_right_price  # features/get_price.feature:3
    Given 1                   # features/steps/BDD_testing_case.py:5
    When right                # features/steps/BDD_testing_case.py:11
    Then roots                # features/steps/BDD_testing_case.py:17

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