Московский государственный технический университет им. Н.Э. Баумана.
Факультет «Информатика и управление»
Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python.»

Выполнил:

студент группы ИУ5-51Б

Забелина Варвара

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю.Е.

Подпись и дата

г. Москва, 2020 г.

Задание лабораторной работы

Изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

Задание

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

Решение

Singleton

Файл класса singleton.py

```
class SingletonMeta(type):
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            instance = super().__call__(*args, **kwargs)
            cls._instances[cls] = instance
        return cls._instances[cls]


class Singleton(metaclass=SingletonMeta):
    file = "file.txt"
    def return_file(cls):
        f = open(cls.file, 'r')
        line = f.read()
        f.close()
        return line

    def write_file(cls, l):
        f = open(cls.file, 'a')
        f.write(l)
        f.close()
```

Файл тестов main.py

```
from singleton.singletonClass import Singleton
import unittest
```

```
class TestsSingleton(unittest.TestCase):
    def test_singleton_return_file(self):
        singleton = Singleton()
        sfile = singleton.return_file()

        file = open(singleton.file, 'r')
        expect = file.read()
        file.close()
        self.assertEqual(sfile, expect)

    def test_singleton(self):
        s1 = Singleton()
        s2 = Singleton()
        self.assertEqual(id(s1), id(s2))

    def test_singleton_write_file(self):
        singleton = Singleton()
        line = "Хлопья летят наверх"
        singleton.write_file(line)

        file = open(singleton.file, 'r')
        fileText = file.read()
        file.close()

        expect = ""
        for i in range(len(line)):
            expect += fileText[len(fileText) - len(line) + i]
        self.assertEqual(line, expect)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Facade

Файл класса facade.py

```
from __future__ import annotations

class Facade:

    def __init__(self, numbers: list) -> None:
        self._sum = Sum(numbers)
        self._div = Division()

    def operation(self) -> str:
        results = []

        num = self._sum.operation_plus()
        den = self._sum.return_amount()
        self._div.set_variables(num, den)
        result = self._div.division()
```

```
        results.append("Среднее арифметическое чисел: ")
        results.append(self._sum.list_items())
        results.append("Результат: ")
        results.append(str(result))
        return "\n".join(results)


class Sum:
    _list = []

    def __init__(self, numbers: list):
        self._list = numbers

    def operation_plus(self):
        sum = 0
        for i in self._list:
            sum += i
        return sum

    def return_amount(self):
        return len(self._list)

    def list_items(self) -> str:
        return str(self._list)


class Division:

    def __init__(self, num=1.0, den=1.0):
        self._numerator = num
        self._denominator = den

    def division(self):
        return self._numerator / self._denominator

    def set_variables(self, n, d):
        self._denominator = d
        self._numerator = n
```

Файл тестов main.py

```
from facade.facadeClass import Facade, Sum, Division
import unittest


class TestsFacade(unittest.TestCase):

    def test_facade_operation(self):
        for i in range(3, 10):
            items = []
            for j in range(20):
                items.append(j)
            with self.subTest(items=items):
```

```
                f = Facade(items)
                result = f.operation()

                sum = Sum(items)
                num = sum.operation_plus()
                den = sum.return_amount()
                div = Division(num, den)
                res = div.division()

                sresult = []
                sresult.append("Среднее арифметическое чисел: ")
                sresult.append(str(items))
                sresult.append("Результат: ")
                sresult.append(str(res))
                expected = "\n".join(sresult)
                self.assertEqual(result, expected)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Command

Файл класса command.py

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


class Command(ABC):
    @abstractmethod
    def execute(self) -> None:
        pass


class Order(Command):
    def __init__(self, payload: list) -> None:
        self.orders = payload

    def execute(self) -> None:
        print(f"Ваш заказ: "
              f"({self.orders})")

    def return_guest_order(self):
        return self.orders


class Cooking(Command):
    def __init__(self, receiver: Receiver, l: list) -> None:
        self._receiver = receiver
        self._list_of_orders = l

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

```
        for o in self._list_of_orders:
            self._receiver.start_cook(o)
        for o in self._list_of_orders:
            self._receiver.ready(o)


class Receiver:
    def start_cook(self, a: str) -> None:
        print(f"Ваш {a} начали готовить.\n", end="")

    def ready(self, b: str) -> None:
        print(f"Ваш {b} готово.\n", end="")


class Invoker:
    _on_start = None
    _on_finish = None

    def set_on_start(self, command: Command):
        self._on_start = command

    def set_on_finish(self, command: Command):
        self._on_finish = command

    def do_something_important(self) -> None:
        print("Здрвастуйте, желаете сделать заказ?\n")
        if isinstance(self._on_start, Command):
            self._on_start.execute()

            print("Передаю ваш заказ на кухню\n")
            if isinstance(self._on_finish, Command):
                self._on_finish.execute()
        else:
            print("Хорошо, я подойду позже\n")
```

Файл работы программы main.py

```
from comand.command import Invoker, Order, Receiver, Cooking

if __name__ == "__main__":
    invoker = Invoker()
    order = ["Макароны по-
флотски", "Стейк средней прожарки", "Клубничный молочный коктейль",
             "Латте на миндальном с маршмеллоу"]
    invoker.set_on_start(Order(order))
    receiver = Receiver()
    invoker.set_on_finish(Cooking(receiver, order))

    invoker.do_something_important()
```

Результат

Singleton

```
test_singleton (__main__.TestsSingleton) ... ok
test_singleton_return_file (__main__.TestsSingleton) ... ok
test_singleton_write_file (__main__.TestsSingleton) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.005s

OK
```

Facade

```
test_facade_operation (__main__.TestsFacade) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Command

```
Здрвастуйте, желаете сделать заказ?

Ваш заказ: (['Макароны по-флотски', 'Стейк средней прожарки', 'Клубничный молочный коктейль', 'Латте на миндальном с маршмеллоу'])
Передаю ваш заказ на кухню

Ваш Макароны по-флотски начали готовить.
Ваш Стейк средней прожарки начали готовить.
Ваш Клубничный молочный коктейль начали готовить.
Ваш Латте на миндальном с маршмеллоу начали готовить.
Ваш Макароны по-флотски готово.
Ваш Стейк средней прожарки готово.
Ваш Клубничный молочный коктейль готово.
Ваш Латте на миндальном с маршмеллоу готово.
```