**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: Проверил:

студент группы ИУ5-51Б преподаватель каф. ИУ5

Ахвердиев Валерий Гапанюк Ю.Е.

Подпись и дата: 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("Здрваствуйте, 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))

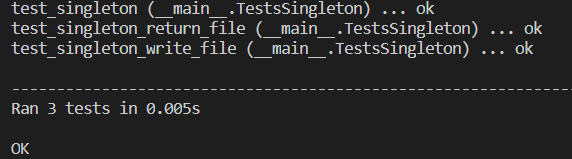
    receiver = Receiver()

    invoker.set\_on\_finish(Cooking(receiver, order))

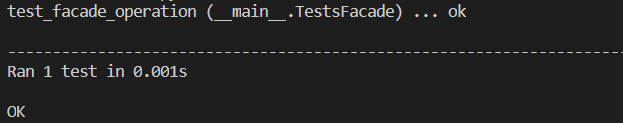
    invoker.do\_something\_important()

# Результат

## Singleton



## Facade



## Command

