**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Системы обработки информации и управления»

Кафедра ИУ5. Курс «РИП»

Отчет по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-53 |  | преподаватель каф. ИУ5 |
| Шамаилов М. Т. |  | Гапанюк Ю. Е. |
| Подпись и дата: 28.12.2020 |  | Подпись и дата: |

Москва, 2020 г.

Оглавление

[1. Цель лабораторной работы 2](#_Toc60076629)

[2. Задание 2](#_Toc60076630)

[3. Исходный код программы 3](#_Toc60076631)

[3.1. Файл *adapter.py* 3](#_Toc60076632)

[3.2. Файл *observer.py* 4](#_Toc60076633)

[3.3. Файл *test\_observers.py* 6](#_Toc60076634)

[4. Результаты работы программы 7](#_Toc60076635)

[4.1. Файл *adapter.py* 7](#_Toc60076636)

[4.2. Файл *observer.py* 7](#_Toc60076637)

[4.3. Файл *test-observers.py* 7](#_Toc60076638)

[5. Github — репозиторий 7](#_Toc60076639)

1. Цель лабораторной работы

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

1. Задание
2. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать [следующий каталог.](https://refactoring.guru/ru/design-patterns/catalog)
3. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.
4. Исходный код программы

# 3.1. Файл *adapter.py*

#*Структурный паттерн Adapter*

class Old:

    def *get*(self):

*return* "1234"

class New:

    def *get\_integer\_type*(self):

*return* 456

class Adapter(New):

    """

    Создал сам паттерн

    (в качестве первого взял простейший адаптер)

    """

    def *get*(self):

*return* str(self.get\_integer\_type())

def *main*(obj):

    print("Результат: " + obj.get())

*if* \_\_name\_\_ == '\_\_main\_\_':

    obj = Adapter()

    main(obj)

# 3.2. Файл *observer.py*

#*Поведенческий паттерн Observer*

#*Реализация на примере паблика в социальной сети*

*from* abc *import* ABC, abstractmethod

class Observer(ABC):

    #*\*\*\* интерфейс подписчиков*

    @*abstractmethod*

    def *update*(self, message: str) -> None:

*pass*

class Memodel(ABC):

    """

    Тут будут авторы публикаций в паблике

    """

    def \_\_init\_\_(self) -> None:

        self.observers = []

    def *subscribe\_new\_follower*(self, observer: Observer) -> None:

        print (observer.name + " присоединился к нашей команде!")

        self.observers.append(observer)

    def *unsubscribe*(self, observer):

        print (observer.name + " покинул нас... =(")

        self.observers.remove(observer)

    def *talk\_and\_send\_message\_to\_other\_followers*(self, message: str) -> None:

*for* observer *in* self.observers:

            print(observer.update(message))

class Publication(Memodel):

    """

    Тут будет какое-то конкретное обсуждение публикации/записи

    """

    def *add\_news*(self, posts: str) -> None:

        self.talk\_and\_send\_message\_to\_other\_followers(posts)

class Follower(Observer):

    """

    А тут сидит подписчик, следящий за обновлением группы

    """

    def \_\_init\_\_(self, name: str) -> None:

        self.name = name

    def *update*(self, message: str):

*return* f'{self.name} получил уведомление: {message}'

#*главная функция в лабе*

def *main*():

    post = Publication()

    is\_follower\_1=Follower('sky\_active')

    is\_follower\_2=Follower('markant')

    post.subscribe\_new\_follower(is\_follower\_1)

    post.subscribe\_new\_follower(is\_follower\_2)

    post.add\_news('Наблюдатель - поведенческий шаблон проектирования')

    post.unsubscribe(is\_follower\_1)

*if* \_\_name\_\_ == '\_\_main\_\_':

    main()

# 3.3. Файл *test\_observers.py*

#*реализуем технологию тестирования  tdd на примере паттерна Observer*

#*! сначала импортнём все, что нужно*

*import* unittest

*from* observer *import* Follower

*from* observer *import* Memodel

class ObserverTestCase(unittest.TestCase):

    """

    Главный тестовый класс

    """

    #*этот небольшой минитест должен проверять добавился ли новый подписчик к нам в паблик*

    def *subscribe\_new\_follower\_minitest*(self):

        follower\_subscribed\_nikname = Follower("Петя")

        post = Memodel()

        post.subscribe\_new\_follower(follower\_subscribed\_nikname)

        self.assertEqual(type(follower\_subscribed\_nikname), type(post.observers[0]))

      #*а этот небольшой минитест должен проверять отписался ли подписчик щт паблика*

    def *test\_unsubscribe*(self):

        foll1 = Follower("Петя")

        post = Memodel()

        post.subscribe\_new\_follower(foll1)

        post.unsubscribe(foll1)

        self.assertEqual(0, len(post.observers))

    def *test\_for\_users\_and\_followers*(self):

        post = Memodel()

        foll1 = Follower("Петя")

        message='Наблюдатель - поведенческий шаблон проектирования'

        self.assertEqual( f'Петя получил уведомление: {message}',

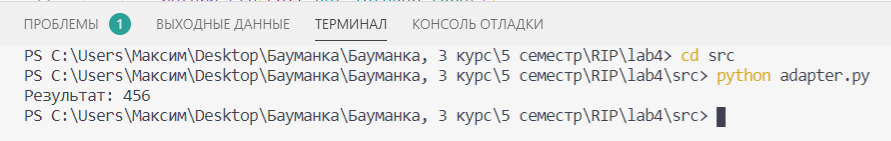
                         foll1.update(message))

*if* \_\_name\_\_ == '\_\_main\_\_':

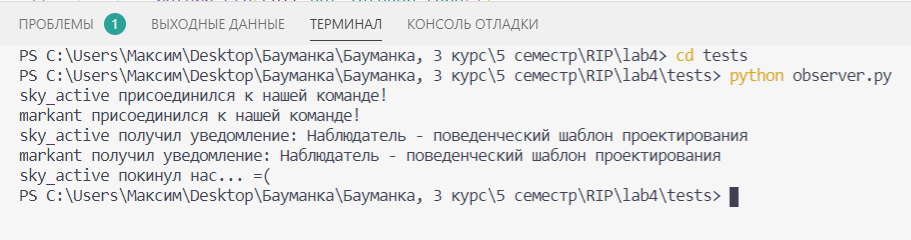
        unittest.main()

1. Результаты работы программы

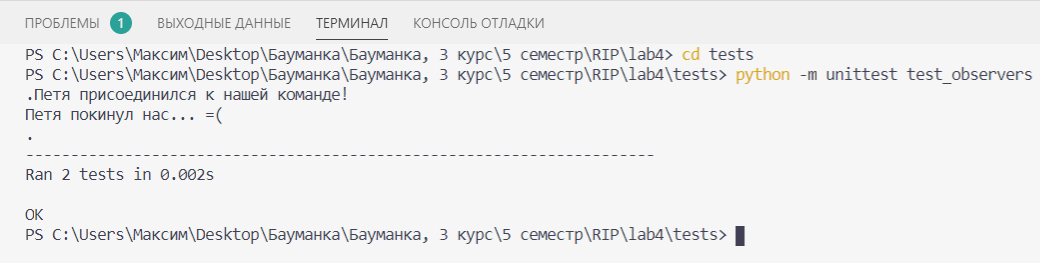
# 4.1. Файл *adapter.py*



# 4.2. Файл *observer.py*



# 4.3. Файл *test-observers.py*



1. Github — репозиторий

github.com/ShamailovMax/RIP/tree/master/lab4