**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет ««Информатика и системы управления»

# Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №4

# «Шаблоны проектирования и модули тестирования в Python»

Вариант 8

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35Б |  |  |
| Костерин А.С. |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г

**Постановка задачи:**

**Цель лабораторной работы:** изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

### 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.](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Выполнение:**

Файл – adapter.py

class OldPrinter:

    def print\_old(self, message: str):

        return f"OldPrinter: {message}"

class PrinterAdapter:

    def \_\_init\_\_(self, old\_printer: OldPrinter):

        self.old\_printer = old\_printer

    def print(self, message: str):

        return self.old\_printer.print\_old(message)

if \_\_name\_\_ == "\_\_main\_\_":

    old\_printer = OldPrinter()

    printer\_adapter = PrinterAdapter(old\_printer)

    print(printer\_adapter.print("Hello"))

Файл – bdd.py

from behave import given, when, then

from fabric import AnimalFactory

@given('an animal factory')

def step\_given\_factory(context):

    context.factory = AnimalFactory()

@when('I create a dog')

def step\_create\_dog(context):

    context.animal = context.factory.create\_animal("dog")

@when('I create a cat')

def step\_create\_cat(context):

    context.animal = context.factory.create\_animal("cat")

@then('the sound should be "{sound}"')

def step\_check\_sound(context, sound):

    assert context.animal.sound() == sound

Файл – fabric.py

from abc import ABC, abstractmethod

class Animal(ABC):

    @abstractmethod

    def sound(self) -> str:

        pass

class Dog(Animal):

    def sound(self) -> str:

        return "Woof!"

class Cat(Animal):

    def sound(self) -> str:

        return "Meow!"

class AnimalFactory:

    @staticmethod

    def create\_animal(animal\_type: str) -> Animal:

        if animal\_type == "dog":

            return Dog()

        elif animal\_type == "cat":

            return Cat()

        else:

            raise ValueError(f"Unknown animal type: {animal\_type}")

if \_\_name\_\_ == "\_\_main\_\_":

    factory = AnimalFactory()

    dog = factory.create\_animal("dog")

    cat = factory.create\_animal("cat")

    print(dog.sound())

    print(cat.sound())

Файл – mock.py

from unittest.mock import MagicMock

from observer import Subject

def test\_observer\_mock():

    subject = Subject()

    observer\_mock = MagicMock()

    subject.attach(observer\_mock)

    subject.set\_state("new state")

    observer\_mock.update.assert\_called\_with("new state")

Файл – observer.py

class Subject:

    def \_\_init\_\_(self):

        self.\_observers = []

        self.\_state = None

    def attach(self, observer):

        self.\_observers.append(observer)

    def notify(self):

        for observer in self.\_observers:

            observer.update(self.\_state)

    def set\_state(self, state):

        self.\_state = state

        self.notify()

class Observer:

    def update(self, state):

        print(f"Observer: Reacted to the state change: {state}")

if \_\_name\_\_ == "\_\_main\_\_":

    subject = Subject()

    observer1 = Observer()

    observer2 = Observer()

    subject.attach(observer1)

    subject.attach(observer2)

    subject.set\_state("State 1")

    subject.set\_state("State 2")

Файл – test.py

import unittest

from fabric import AnimalFactory

class TestAnimalFactory(unittest.TestCase):

    def test\_dog\_creation(self):

        factory = AnimalFactory()

        dog = factory.create\_animal("dog")

        self.assertEqual(dog.sound(), "Woof!")

    def test\_cat\_creation(self):

        factory = AnimalFactory()

        cat = factory.create\_animal("cat")

        self.assertEqual(cat.sound(), "Meow!")

    def test\_invalid\_animal(self):

        factory = AnimalFactory()

        with self.assertRaises(ValueError):

            factory.create\_animal("fish")

if \_\_name\_\_ == "\_\_main\_\_":

    unittest.main()

Пример выполнения:







