**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
«Модульное тестирование в Python»

Выполнил:
студент группы ИУ5-36Б
Ковалев Е.А.

Проверил:
Преподаватель каф. ИУ5
Гапанюк Ю.Е.

2024 г.

# Описание задания

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. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

# Текст программы

```python
# factory_method.py
# Реализует порождающий шаблон - Фабричный метод

from abc import ABC, abstractmethod

class Product(ABC):
    @abstractmethod
    def get_desc(self):
        pass

class Bullet(Product):
    def get_desc(self):
        return "Произведен патрон"

class Cigarette(Product):
    def get_desc(self):
        return "Произведена сигарета"

class Factory(ABC):
    @abstractmethod
    def create_product(self):
        pass

class BulletFactory(Factory):
    def create_product(self):
        return Bullet()

class CigaretteFactory(Factory):
    def create_product(self):
        return Cigarette()
```

```python
# test_factory_method.py
# TDD-тестирование для паттерна фабричный метод

import unittest
from factory_method import BulletFactory, CigaretteFactory

class TestFactoryMethod(unittest.TestCase):
    def test_factory_method(self):
        bullet_factory = BulletFactory()
        cigarette_factory = CigaretteFactory()

        bullet = bullet_factory.create_product()
        self.assertEqual(bullet.get_desc(), "Произведен патрон")

        cigarette = cigarette_factory.create_product()
        self.assertEqual(cigarette.get_desc(), "Произведена сигарета")

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

```python
# composite.py
# Реализует структурный паттерн - Компоновщик

class Product:
    def get_desc(self):
        pass

class SingularProduct(Product):
    def __init__(self, name):
        self.name = name

    def get_desc(self):
        return self.name

class ProductPack(Product):
    def __init__(self, name):
        self.name = name
        self._components = []

    def add(self, product: Product):
        self._components.append(product)

    def get_desc(self):
        descs = [component.get_desc() for component in self._components]
        return f"В пачке: {self.name}, находятся: {', '.join(descs)}"
```

```python
# test_composite.py
# BDD-тестирование для паттерна Компоновщик

import pytest
from pytest_bdd import scenario, given, when, then
from composite import SingularProduct, ProductPack

@pytest.fixture
def product():
    return SingularProduct("Сигарета 'Беломорканал'")

@pytest.fixture
def pack():
    return ProductPack("Пачка сигарет")

@scenario('composite.feature', 'Упаковка пачки')
def test_composite():
    pass

@given('одна сигарета')
def product_fixture(product):
    return product

@given('пачка сигарет')
def pack_fixture(pack):
```

```
    return pack

@when('сигарета кладется в пачку')
def cig2pack(product, pack):
    pack.add(product)

@then('сигарета отображается в составе данной пачки')
def check_pack(pack):
    assert pack.get_desc() == "В пачке: Пачка сигарет, находятся: Сигарета 'Беломорканал'"
```

**#composite.Feature**

**Feature: Тестирование паттерна Компоновщик**

**Scenario: Упаковка пачки**
**Given одна сигарета**
**And пачка сигарет**
**When сигарета кладется в пачку**
**Then сигарета отображается в составе данной пачки**

```
# strategy.py
# Реализует повededенческий шаблон - стратегия

from abc import ABC, abstractmethod

class DeliveryStrategy(ABC):
    @abstractmethod
    def deliver(self, order):
        pass

class StandardDelivery(DeliveryStrategy):
    def deliver(self, order):
        return f"Доставляю {order} стандартной доставкой."

class ExpressDelivery(DeliveryStrategy):
    def deliver(self, order):
        return f"Доставляю {order} экспресс-доставкой."

class Order:
    def __init__(self, delivery_strategy: DeliveryStrategy):
        self.delivery_strategy = delivery_strategy

    def set_delivery_strategy(self, delivery_strategy: DeliveryStrategy):
        self.delivery_strategy = delivery_strategy

    def deliver_order(self, order):
        return self.delivery_strategy.deliver(order)
```

```
# test_strategy.py
# Тестирование паттерна Стратегия с помощью Mock-объектов

import unittest
from unittest.mock import Mock
from strategy import Order, StandardDelivery, ExpressDelivery
```

```python
class TestStrategy(unittest.TestCase):
    def test_delivery_strategy(self):
        standard_delivery = Mock(spec=StandardDelivery)
        express_delivery = Mock(spec=ExpressDelivery)

        order = Order(standard_delivery)

        order.deliver_order("Заказ #123")

        standard_delivery.deliver.assert_called_with("Заказ #123")

        order.set_delivery_strategy(express_delivery)

        order.deliver_order("Заказ #124")
        express_delivery.deliver.assert_called_with("Заказ #124")

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

# Экранные формы с примерами выполнения программы