**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 №2

«Объектно-ориентированные возможности языка Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Маркин Денис |  | Нардид Анатолий Николаевич |
|  |  |  |
|  |  |  |

Москва, 2024 г.

Постановка задачи

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. 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

Behavioral.py

from abc import ABC, abstractmethod

#State Шаблон состояния

class State(ABC):

    @abstractmethod

    def handle(self):

        pass

class Outside(State):

    def handle(self):

        return "Студент не в ВУЗе"

class Inside(State):

    def handle(self):

        return "Студент учится"

class Walk(State):

    def handle(self):

        return "Студент в пути"

class Window(State):

    def handle(self):

        return "Студент в ВУЗе, но сейчас у него окно"

class Student:

    def \_\_init\_\_(self):

        self.state = Outside()

    def set\_state(self, state: State):

        self.state = state

    def where(self):

        return self.state.handle()

me = Student()

print(me.where())

me.set\_state(Inside())

print(me.where())

me.set\_state(Window())

print(me.where())

me.set\_state(Outside())

print(me.where())

me.set\_state(Walk())

print(me.where())

Creational.py

#Singleton Порождающий шаблон

class Shop:

    instance = None #Unity moment

    def \_\_new\_\_(cls):

        if cls.instance is None:

            cls.instance = super(Shop, cls).\_\_new\_\_(cls)

            cls.instance.products = []

        return cls.instance

    def add\_product(self, \_p):

        self.products.append(\_p)

    def get\_products(self):

        return self.products

# Пример использования

shop = Shop()

shop.add\_product("Bananas")

shop.add\_product("Apples")

another\_shop = Shop()

another\_shop.add\_product("Oranges")

print(shop.get\_products())

print(another\_shop.get\_products())

print("equal" if another\_shop == shop else "mistake!")

Struct.py

from abc import ABC, abstractmethod

#Bridge Структурный

class Color(ABC): #Абстракция

    @abstractmethod

    def get\_color(self):

        pass

class Red(Color): #Реализация

    def get\_color(self):

        return "красный"

class Blue(Color): #Реализация

    def get\_color(self):

        return "синий"

class Shape(ABC): #Абстракция с ссылкой на реализацию

    def \_\_init\_\_(self, color: Color):

        self.color = color

    @abstractmethod

    def info(self):

        pass

class Square(Shape): #Ну дальше ты пон

    def info(self):

        return f"Квадрат, цвет: {self.color.get\_color()}"

class Circle(Shape):

    def info(self):

        return f"Окружность, цвет: {self.color.get\_color()}"

red = Red()

blue = Blue()

square\_red = Square(red)

circle\_blue = Circle(blue)

print(square\_red.info())

print(circle\_blue.info())

test\_Behavioral.py

import unittest

# Тест пройден Сначала тесты, потом разработка

from Behavioral import Student, Outside, Inside, Window, Walk

class TestStudent(unittest.TestCase):

    def setUp(self):

        self.student = Student()

    def test\_initial\_state(self):

        self.assertEqual(self.student.where(), "Студент не в ВУЗе")

    def test\_state\_inside(self):

        self.student.set\_state(Inside())

        self.assertEqual(self.student.where(), "Студент учится")

    def test\_state\_window(self):

        self.student.set\_state(Window())

        self.assertEqual(self.student.where(), "Студент в ВУЗе, но сейчас у него окно")

    def test\_state\_outside(self):

        self.student.set\_state(Outside())

        self.assertEqual(self.student.where(), "Студент не в ВУЗе")

    def test\_state\_walk(self):

        self.student.set\_state(Walk())

        self.assertEqual(self.student.where(), "Студент в пути")

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

test\_Creational.py

import unittest

from unittest.mock import patch

from Creational import Shop

class TestShopSingleton(unittest.TestCase):

    @patch.object(Shop, 'add\_product')

    def test\_add\_product\_using\_mock(self, mock\_add\_product):

        shop = Shop()

        mock\_add\_product("Bananas")

        mock\_add\_product("Apples")

        self.assertEqual(mock\_add\_product.call\_count, 2)

        shop.add\_product("Bananas")

        shop.add\_product("Apples")

        self.assertIn("Bananas", shop.get\_products())

        self.assertIn("Apples", shop.get\_products())

    def test\_singleton\_property(self):

        shop1 = Shop()

        shop2 = Shop()

        shop1.add\_product("Grapes")

        self.assertIs(shop1, shop2)

        self.assertIn("Grapes", shop2.get\_products())

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

features/steps/steps.py

from behave import given, when, then

from Struct import Red, Blue, Square, Circle

@given('I have a red color')

def step\_given\_red\_color(context):

    context.color = Red()

@when('I create a square with that color')

def step\_when\_create\_square(context):

    context.shape = Square(context.color)

@then('it should return "{expected\_output}"')

def step\_then\_check\_output(context, expected\_output):

    assert context.shape.info() == expected\_output

Результаты работы программы

 Struct.py

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описаниеCreational.py

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описаниеBehavioral.py

Изображение выглядит как текст, снимок экрана, Шрифт, инструмент

Автоматически созданное описание

Test\_Behavioral.py

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Test\_Creational.py

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

features