**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 №5-6

«Модульное тестирование в Python»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Багин М. В. |  | Гапанюк Ю. Е. |
|  |  |  |
|  |  |  |

2024 г.

### Задание:

1. Выберите любой фрагмент кода из лабораторных работ 1 или 2 или 3-4.
2. Модифицируйте код таким образом, чтобы он был пригоден для модульного тестирования.
3. Разработайте модульные тесты. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк (не менее 3 тестов).
   * BDD - фреймворк (не менее 3 тестов).
   * Создание Mock-объектов (необязательное дополнительное задание).

**Текст программы**

singletone\_meta.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]

adapter.py

class TrafficLightAdapter:

def \_\_init\_\_(self, traffic\_light):

self.traffic\_light = traffic\_light

def switch(self):

self.traffic\_light.change\_state()

def current\_light(self):

return self.traffic\_light.get\_state()

state.py

class State:

def next(self, context):

raise NotImplementedError

def \_\_str\_\_(self):

raise NotImplementedError

class Red(State):

def next(self, context):

context.set\_state(Yellow())

print("Switching from RED to YELLOW")

def \_\_str\_\_(self):

return "RED"

class Yellow(State):

def next(self, context):

context.set\_state(Green())

print("Switching from YELLOW to GREEN")

def \_\_str\_\_(self):

return "YELLOW"

class Green(State):

def next(self, context):

context.set\_state(Red())

print("Switching from GREEN to RED")

def \_\_str\_\_(self):

return "GREEN"

traffic\_light.py

from state import Red

from singleton\_meta import SingletonMeta

class TrafficLight(metaclass=SingletonMeta):

def \_\_init\_\_(self):

self.state = Red()

def set\_state(self, state):

self.state = state

def change\_state(self):

self.state.next(self)

def get\_state(self):

return str(self.state)

test\_traffic\_light.py

import unittest

from unittest.mock import patch

from traffic\_light import TrafficLight

from adapter import TrafficLightAdapter

class TestTrafficLight(unittest.TestCase):

def setUp(self):

self.traffic\_light = TrafficLight()

self.adapter = TrafficLightAdapter(self.traffic\_light)

def test\_singleton(self):

traffic\_light2 = TrafficLight()

self.assertIs(self.traffic\_light, traffic\_light2, "TrafficLight is not a singleton")

def test\_state\_transition(self):

with patch('builtins.print') as mocked\_print:

self.assertEqual(self.traffic\_light.get\_state(), "RED")

self.traffic\_light.change\_state()

mocked\_print.assert\_called\_with("Switching from RED to YELLOW")

self.assertEqual(self.traffic\_light.get\_state(), "YELLOW")

self.traffic\_light.change\_state()

mocked\_print.assert\_called\_with("Switching from YELLOW to GREEN")

self.assertEqual(self.traffic\_light.get\_state(), "GREEN")

self.traffic\_light.change\_state()

mocked\_print.assert\_called\_with("Switching from GREEN to RED")

self.assertEqual(self.traffic\_light.get\_state(), "RED")

def test\_adapter(self):

with patch('builtins.print') as mocked\_print:

self.assertEqual(self.adapter.current\_light(), "RED")

self.adapter.switch()

mocked\_print.assert\_called\_with("Switching from RED to YELLOW")

self.assertEqual(self.adapter.current\_light(), "YELLOW")

**Анализ результатов**

test\_singleton



test\_state\_transition



test\_adapter

