Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Отчёт по Лабораторной работе №4
“Паттерны программирования и тестирование на Python”

Отчёт
(вид документа)

Листы А4
(вид носителя)

9
(количество листов)

Исполнитель: Студент группы ИУ5-54Б
Савельев Алексей
Александрович

Москва - 2020

Цель работы: Изучения реализации шаблонов проектирования и возможностей модульного тестирования Python.

Задание:

- Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.
- Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:

  - TDD - фреймворк.
  - BDD - фреймворк.
  - Создание Mock-объектов.

Листинг программы

Порождающий паттерн «Фабричный метод»

- FabricMethod.py

```
from abc import ABC, abstractmethod

# Паттерн "Фабрика"
class Manufacturer(ABC):

    @abstractmethod
    def toProduce(self):
        pass

    def output(self):
        product = self.toProduce()
        result = [f"Этот производитель производит: {product.info()[0]}",
product.info()[1]]
        return result

class Product(ABC):

    @property
    @abstractmethod
    def movement(self):
        pass

    @property
    @abstractmethod
    def model(self):
        pass

    def info(self) -> []:
        return [f"{self.model}, it's {self.movement}", (self.model,
self.movement)]


class AutoManufacturer(Manufacturer):

    def toProduce(self) -> Product:
        return Auto()
```

```
class Auto(Product):

    @property
    def movement(self) -> str:
        return "ездиет"

    @property
    def model(self) -> str:
        return "Inspire350"

class PlaneManufacturer(Manufacturer):

    def toProduce(self) -> Product:
        return Plane()

class Plane(Product):

    @property
    def movement(self) -> str:
        return "летает"

    @property
    def model(self) -> str:
        return "Dreamweawer770"

def clientCode1(m: Manufacturer):

    return m.output()


if __name__ == "__main__":
    print("\033[36mПАТТЕРН 'ФАБРИКА'\033[0m")
    print("\033[32mApp: Запущенно с AutoManufacturer.\033[0m")
    print("\033[33mКлиент: Я не щарю за класс Производителя, но это все еще
работает...\033[0m")
    print(clientCode1(AutoManufacturer())[0])
    print()

    print("\033[32mApp: Запущенно с PlaneManufacturer.\033[0m")
    print("\033[33mКлиент: Я не щарю за класс Производителя, но это все еще
работает...\033[0m")
    print(clientCode1(PlaneManufacturer())[0])
```

Структурный паттерн «Адаптер»

- AdapterPattent.py

```
# Паттерн "Адаптер"
class Strait:
    def __init__(self, number):
        self.straitNumber = int(number)

    def straitOutput(self):
        return self.straitNumber


class Bin:
    def __init__(self, number):
        self.binNumber = int(number)

    def binOutput(self):
        return self.binNumber
```

```
class BinPresenter(Strait, Bin):

    def getBinary(self, n):
        n = int(n)
        if (n == 0):
            return "0"
        elif (n == 1):
            return "1"
        else:
            return self.getBinary(int(n / 2)) + str(n % 2)

    def binOutput(self) -> str:
        return self.getBinary(self.straitNumber)


def clientCode2(target: "Bin"):
    """
    Клиентский код поддерживает все классы, использующие интерфейс Bin.
    """
    return target.binOutput()


if __name__ == "__main__":

    print("\033[36mПАТТЕРН 'АДАПТЕР'\033[0m")
    print("\033[33mКлиент: могу работать с бинарными числами:\033[0m")
    target = Bin(1010)
    print(f"Возврат клиентского кода: \033[32m{clientCode2(target)}\033[0m\n")

    adaptee = Strait(12345)
    print("\033[33mКлиент: класс Adaptee имеет не подходящий мне
интерфейс\033[0m")
    print(f"Вывод класса Adaptee: \033[31m{adaptee.straitOutput()}\n\033[0m")

    print("\033[33mКлиент: но я могу работать с этим классом через
адаптер:\033[0m")
    adapter = BinPresenter(12345)
    print(f"Неудобное значение: \033[31m{adapter.straitOutput()}\033[0m")
    print(f"Возврат клиентского кода с использованием Адаптера:
\033[32m{clientCode2(adapter)}\033[0m")
```

Поведенческий паттерн «Состояние»

- StatePattern.py

```
import time

class ComputerState(object):
    name = "state"
    allowed = []

    def switch(self, state):
        """ Switch to new state """
        success = False
        if state.name in self.allowed:
            print('\033[33mТекущее стостояние:\033[0m', self, ' =>
\033[32mизменено на \033[35m', state.name, "\033[0m")
            self.__class__ = state
            success = True
        else:
            print('\033[33mТекущее стостояние:\033[0m', self, ' => \033[31m
переключение на \033[35m', state.name, '\033[31mневозможно.\033[0m')
        return success
```

```
    def __str__(self):
        return self.name


class Off(ComputerState):
    name = "off"
    allowed = ['on']


class On(ComputerState):
    """ State of being powered on and working """
    name = "on"
    allowed = ['off', 'suspend', 'hibernate']


class Suspend(ComputerState):
    """ State of being in suspended mode after switched on """
    name = "suspend"
    allowed = ['on']


class Hibernate(ComputerState):
    """ State of being in hibernation after powered on """
    name = "hibernate"
    allowed = ['on']


class Computer(object):
    """ A class representing a computer """

    def __init__(self, model='Dell'):
        self.model = model
        # State of the computer - default is off.
        self.state = Off()

    def change(self, state):
        """ Change state """
        prevState = self.state.name
        success = self.state.switch(state)
        if prevState != "off" and success == True:
            tok = time.perf_counter()
            self.t = f"{tok - self.tic:0.4f}"
            print(f"Прошедшее время в состоянии \033[35m{prevState}\033[0m:
{self.t} sec\n")
        if success == True:
            self.tic = time.perf_counter()


if __name__ == "__main__":
    print("\033[36mПАТТЕРН 'СОСТОЯНИЕ'\033[0m")
    comp = Computer()
    comp.change(On)
    time.sleep(3)
    comp.change(Off)
    comp.change(On)
    time.sleep(0.5)
    comp.change(Suspend)
    time.sleep(2)
    comp.change(Hibernate)
    comp.change(On)
    time.sleep(1)
    comp.change(Hibernate)
```

```
    time.sleep(4)
    comp.change(On)
    time.sleep(1)
    comp.change(Off)
```

Тестирование tdd

- tdd.py

```
import unittest
import FabricMethod, AdapterPattern

class TestStringMethods(unittest.TestCase):

    def testFabric1(self):
        returned = FabricMethod.clientCode1(FabricMethod.AutoManufacturer())[1][1]
        self.assertEqual(returned, "ездиет")

    def testFabric2(self):
        returned =
FabricMethod.clientCode1(FabricMethod.PlaneManufacturer())[1][0]
        self.assertEqual(returned, "Dreamweawer770")

    def testAdapter1(self):
        adapter = AdapterPattern.BinPresenter(127)
        returned = AdapterPattern.clientCode2(adapter)
        self.assertEqual(returned, "1111111")

    def testAdapter2(self):
        adapter = AdapterPattern.BinPresenter(1024)
        returned = AdapterPattern.clientCode2(adapter)
        self.assertEqual(returned, "10000000000")

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

Тестирование bdd

- bdd.py

```
import pytest
from pytest import fixture
from pytest_bdd import *
import StatePattern

@pytest.fixture
def computer():
    return StatePattern.Computer()


@scenario('state.feature', 'On the computer')
def test_On_computer():
    pass

@given("Computer is Off")
def comuterIsOff(computer):
    computer.state.name == "off"

@when("Switching on")
def switchingOn(computer):
    computer.change(StatePattern.On)

@then("State should be On")
def no_error_message(computer):
```

```
    assert computer.state.name == "on"

@scenario('state.feature', 'Suspend the computer')
def test_Suspend_computer():
    pass

@given("Computer is On")
def comuterIsOff(computer):
    computer.state.name == "on"

@when("Switching Suspend")
def switchingOn(computer):
    computer.change(StatePattern.On)
    computer.change(StatePattern.Suspend)

@then("State should be Suspend")
def no_error_message(computer):
    assert computer.state.name == "suspend"

@scenario('state.feature', 'Hibirnate the computer')
def test_Hibirnate_computer():
    pass

@given("Computer is Suspended")
def comuterIsOff(computer):
    computer.state.name == "on"

@when("Switching Hibirnate")
def switchingOn(computer):
    computer.change(StatePattern.On)
    computer.change(StatePattern.Suspend)
    computer.change(StatePattern.Hibernate)

@then("State should be Suspend")
def no_error_message(computer):
    assert computer.state.name == "suspend"
```

- state.feature

```
Feature: State switching
    realization of switching different states of the computer

    Scenario: On the computer
        Given Computer is Off

        When Switching on

        Then State should be On

    Scenario: Suspend the computer
        Given Computer is On

        When Switching Suspend

        Then State should be Suspend

    Scenario: Hibirnate the computer
        Given Computer is Suspended

        When Switching Hibirnate

        Then State should be Suspend
```

Результат в консоли

- FabricMethod.py

```
ПАТТЕРН 'ФАБРИКА'
App: Запущенно с AutoManufacturer.
Клиент: Я не щарю за класс Производителя, но это все еще работает...
Этот производитель производит: Inspire350, it's ездиет

App: Запущенно с PlaneManufacturer.
Клиент: Я не щарю за класс Производителя, но это все еще работает...
Этот производитель производит: Dreamweawer770, it's летает

Process finished with exit code 0
```

- AdapterPattern.py

```
ПАТТЕРН 'АДАПТЕР'
Клиент: могу работать с бинарными числами:
Возврат клиентского кода: 1010

Клиент: класс Adaptee имеет не подходящий мне интерфейс
Вывод класса Adaptee: 12345

Клиент: но я могу работать с этим классом через адаптер:
Неудобное значение: 12345
Возврат клиентского кода с использованием Адаптера: 11000000111001

Process finished with exit code 0
```

- StatePattern.py

```
ПАТТЕРН 'СОСТОЯНИЕ'
Текущее стостояние: off  => изменено на  on
Текущее стостояние: on  => изменено на  off
Прошедшее время в состоянии on: 3.0005 sec

Текущее стостояние: off  => изменено на  on
Текущее стостояние: on  => изменено на  suspend
Прошедшее время в состоянии on: 0.5020 sec

Текущее стостояние: suspend  =>  переключение на  hibernate невозможно.
Текущее стостояние: suspend  => изменено на  on
Прошедшее время в состоянии suspend: 2.0021 sec

Текущее стостояние: on  => изменено на  hibernate
Прошедшее время в состоянии on: 1.0041 sec

Текущее стостояние: hibernate  => изменено на  on
Прошедшее время в состоянии hibernate: 4.0050 sec

Текущее стостояние: on  => изменено на  off
Прошедшее время в состоянии on: 1.0031 sec


Process finished with exit code 0
```

- tdd.py

```
Testing started at 14:24 ...
Launching unittests with arguments python -m unittest



Ran 4 tests in 0.004s

OK

Process finished with exit code 0
```

```
Testing started at 14:24 ...
Launching unittests with arguments python -m unittest



Ran 4 tests in 0.006s

FAILED (failures=1)


10000000001 != 10000000000

Expected :10000000000
Actual   :10000000001
<Click to see difference>
```

- bdd.py

```
(venv) [  2:26PM ]  [ dlnwlkmn@MacBook-Pro-Aleksej:~/Desktop/dia-labs/LR4(masterx) ]
 $ pytest bdd.py
========================================= test session starts ==========================================
platform darwin -- Python 3.8.3, pytest-6.2.1, py-1.10.0, pluggy-0.13.1
rootdir: /Users/dlnwlkmn/Desktop/dia-labs/LR4
plugins: bdd-4.0.2
collected 3 items

bdd.py ...                                                                                      [100%]

========================================== 3 passed in 0.04s ===========================================
(venv) [  2:26PM ]  [ dlnwlkmn@MacBook-Pro-Aleksej:~/Desktop/dia-labs/LR4(masterx) ]
 $
```