**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.](https://refactoring.guru/ru/design-patterns/catalog)

 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

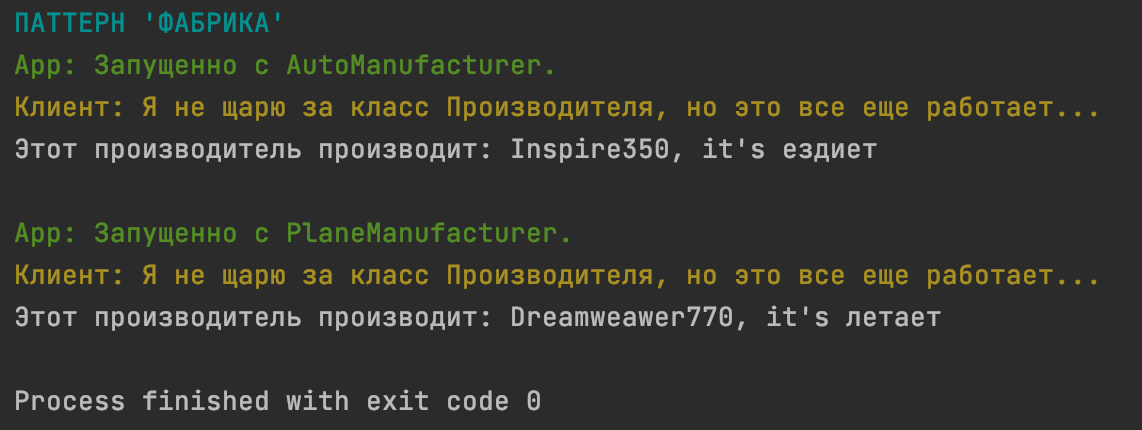
import pytest  
from pytest import fixture  
from pytest\_bdd import \*  
import StatePattern  
  
@pytest.fixture  
def computer():  
 return StatePattern.Computer()  
  
  
@scenario('state.feature', 'On the computer')  
def test\_On\_computer():  
 pass  
  
@given("Computer is Off")  
def comuterIsOff(computer):  
 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

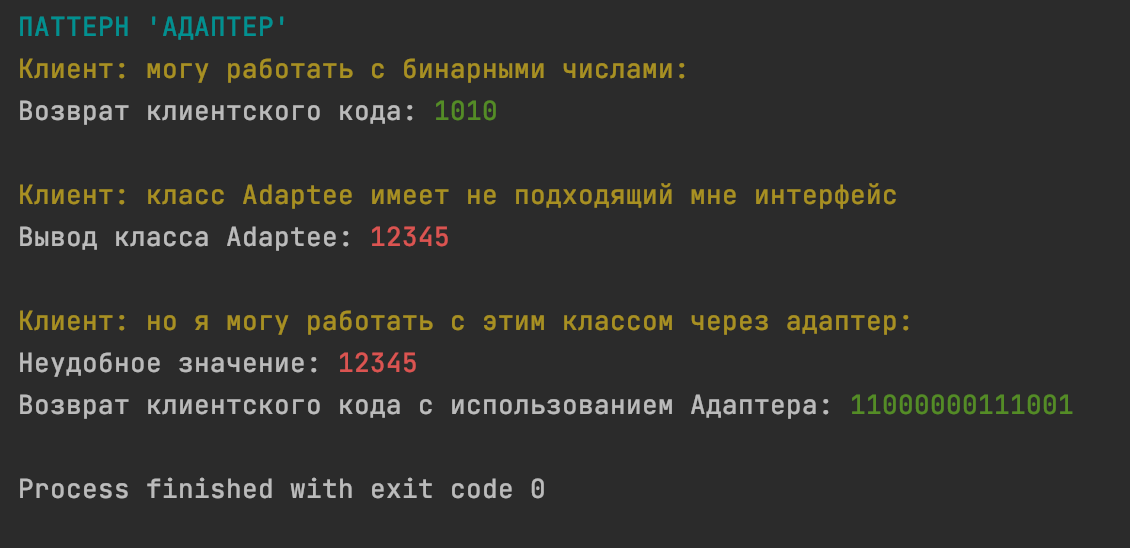
Feature: State switching  
 realization of switching different states of the computer  
  
 Scenario: On the computer  
 Given Computer is Off  
  
 When Switching on  
  
 Then State should be On  
  
 Scenario: Suspend the computer  
 Given Computer is On  
  
 When Switching Suspend  
  
 Then State should be Suspend  
  
 Scenario: Hibirnate the computer  
 Given Computer is Suspended  
  
 When Switching Hibirnate  
  
 Then State should be Suspend

# Результат в консоли

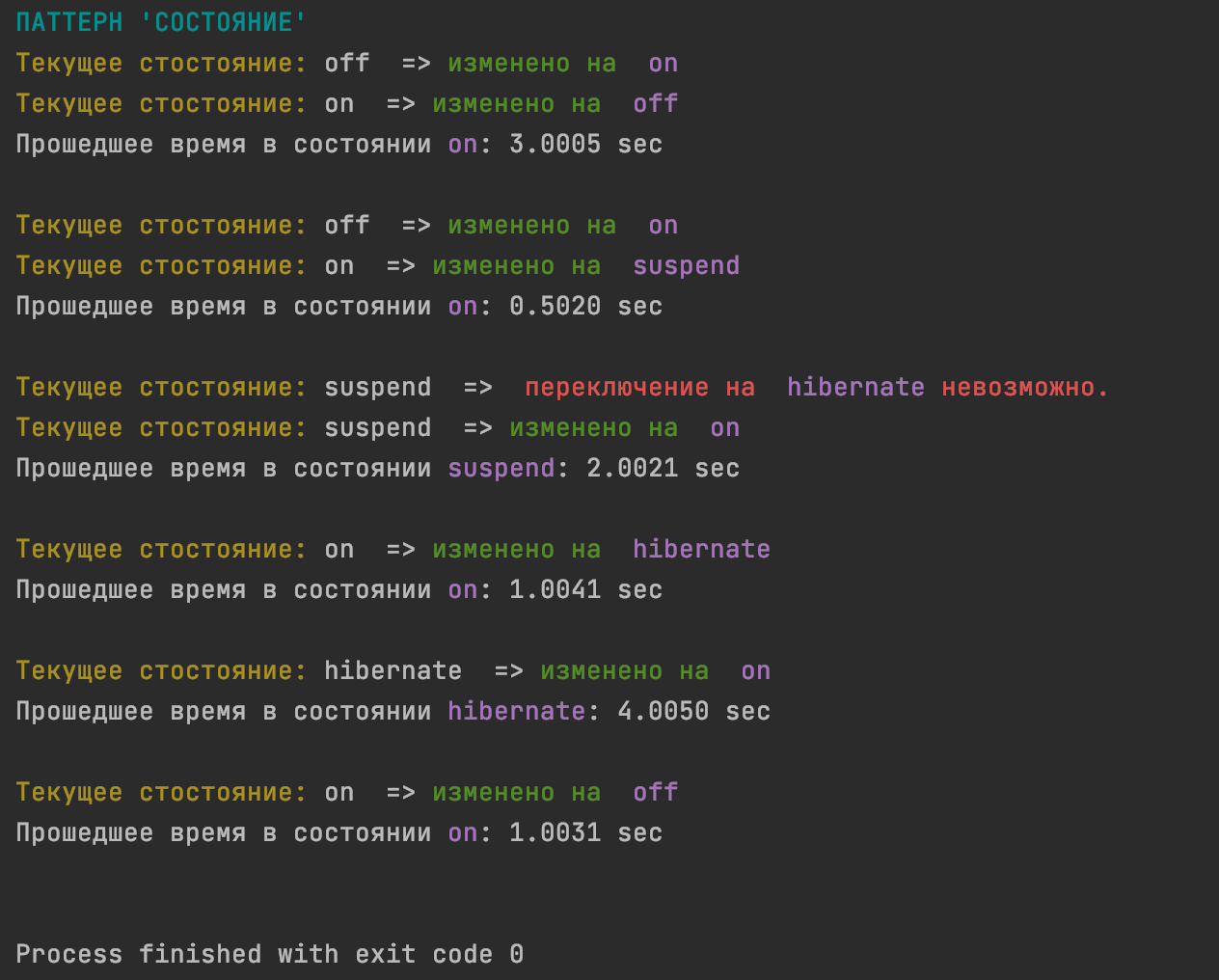
* FabricMethod.py



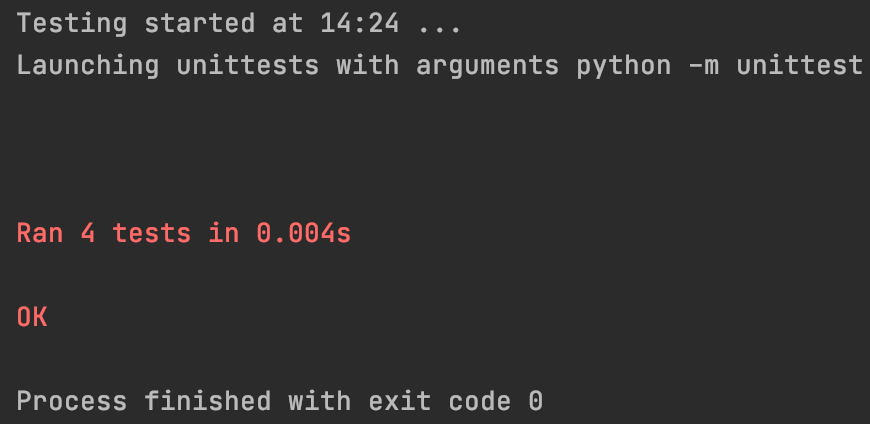
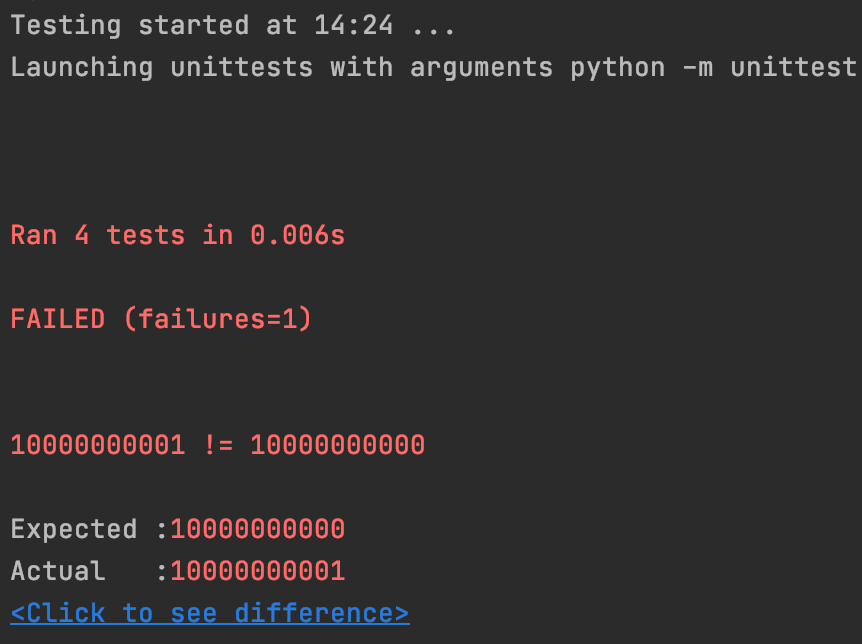
* AdapterPattern.py



* StatePattern.py



* tdd.py

* bdd.py

