Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»
Кафедра «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №4
«Шаблоны проектирования и модульное тестирование в Python»

Выполнил:
студент группы ИУ5-51Б
Алехин Сергей
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

г. Москва, 2020 г.

Задание лабораторной работы

Цель лабораторной работы: изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

Задание:

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

Текст программы

main.py

```
from lab_python_pt.facade.Facade import Facade
def main():
    facade = Facade()
    facade.create_shops(1, 2)
    facade.create_clients(1, 2, 1)
    facade.attach_clients()
    facade.sport  shop  business  logic()
    facade.electronic_shop_business_logic()
    facade.detach  clients()
    facade.sport_shop_business_logic()
    facade.electronic_shop_business_logic()
if __name__ == '__main__':
    main()
```

Observer.py

```
from __future__ import annotations
from abc import ABC, abstractmethod
from random import randrange
from typing import List
class Shops(ABC):
    _SHOP_NAME = None
    def __init__(self, id, count=0):
        self._id = id
        self._count_new_items = count
        self._clients: List[Clients] = []
    @classmethod
    def get_shop_name(cls):
        return cls._SHOP_NAME
    @property
    def id(self):
        return self.  id
    def attach(self, client: Clients) -> None:
        print('{} {}: Attached an observer = {} {}'.format(self._SHOP_NAME, self._id, client.get_client_name(),
                                                           client.id))
        self._clients.append(client)
    def detach(self, client: Clients) -> None:
        print('{} {}: Detached an observer = {} {}'.format(self._SHOP_NAME, self._id, client.get_client_name(),
                                                           client.id))
        self._clients.remove(client)
    def notify(self) -> None:
        print('{} {}: {} observers'.format(self.  SHOP  NAME, self.  id, len(self.  clients)))
        if len(self._clients) != 0:
            print('{} {}: Notifying observers...'.format(self.  SHOP  NAME, self.  id))
            for client in self._clients:
                client.update(self)
    @abstractmethod
    def business_logic(self) -> None:
        pass
```

```
    @property
    def count_new_items(self):
        return self._count_new_items
    @property
    def clients(self):
        return self._clients
class SportShop(Shops):
    _SHOP_NAME = 'SportShop'
    def business_logic(self) -> None:
        if self._count_new_items == 0:
            self._count_new_items = randrange(0, 10)
        print('\n{} {}: I received {} new items'.format(self._SHOP_NAME, self._id, self._count_new_items))
        self.notify()
class ElectronicsShop(Shops):
    _SHOP_NAME = 'ElectronicsShop'
    def business_logic(self) -> None:
        if self._count_new_items == 0:
            self._count_new_items = randrange(0, 15)
        print('\n{} {}: I received {} new items'.format(self._SHOP_NAME, self._id, self._count_new_items))
        self.notify()
class Clients(ABC):
    CLIENT_NAME = None
    def __init__(self, id):
        self._id = id
        self._go_to_shop = False
    @classmethod
    def get_client_name(cls):
        return cls._CLIENT_NAME
    @abstractmethod
    def update(self, shop: Shops) -> None:
        pass
    @property
    def id(self):
        return self._id
    @property
    def go_to_shop(self):
        return self._go_to_shop
class SportShopClient(Clients):
    _CLIENT_NAME = 'SportShopClient'
    def update(self, shop: Shops) -> None:
        self._go_to_shop = False
        if shop.count_new_items >= 5:
            print('{} {}: Reacted to the event'.format(self._CLIENT_NAME, self._id))
            self._go_to_shop = True
class ElectronicsShopClient(Clients):
    _CLIENT_NAME = 'ElectronicsShopClient'
    def update(self, shop: Shops) -> None:
        self._go_to_shop = False
        if shop.count_new_items >= 7:
            print('{} {}: Reacted to the event'.format(self._CLIENT_NAME, self._id))
            self._go_to_shop = True
class SportElectronicsShopClient(Clients):
    _CLIENT_NAME = 'SportElectronicsShopClient'
    def update(self, shop: Shops) -> None:
        self._go_to_shop = False
        if shop.count_new_items >= 5 and shop.get_shop_name() == 'SportShop':
            print('{} {}: Reacted to the event'.format(self._CLIENT_NAME, self._id))
            self._go_to_shop = True
        if shop.count_new_items >= 7 and shop.get_shop_name() == 'ElectronicsShop':
            print('{} {}: Reacted to the event'.format(self._CLIENT_NAME, self._id))
            self._go_to_shop = True
```

ShopFactory.py

```
from __future__ import annotations
from abc import ABC, abstractmethod
from lab_python_pt.observer.Observer import Shops, SportShop, ElectronicsShop
class ShopFactory(ABC):
    _SHOP_FACTORY_NAME = None
    @abstractmethod
```

```
    def factory_method(self, id):
        pass
    @property
    def shop_factory_name(self):
        return self._SHOP_FACTORY_NAME
class SportShopFactory(ShopFactory):
    _SHOP_FACTORY_NAME = 'SportShopFactory'
    def factory_method(self, id) -> Shops:
        print('{}: Create new shop with id = {}'.format(self._SHOP_FACTORY_NAME, id))
        return SportShop(id)
class ElectronicsShopFactory(ShopFactory):
     SHOP  FACTORY  NAME = 'ElectronicsShopFactory'
    def factory_method(self, id) -> Shops:
        print('{}: Create new shop with id = {}'.format(self._SHOP_FACTORY_NAME, id))
        return ElectronicsShop(id)
def get_shop(factory: ShopFactory, id):
    return factory.factory_method(id)
```

ClientsFactory.py

```
from __future__ import annotations
from abc import ABC, abstractmethod
from lab_python_pt.observer.Observer import Clients, SportShopClient, ElectronicsShopClient, SportElectronicsShopClient
class ClientFactory(ABC):
    _CLIENT_FACTORY_NAME = None
    @abstractmethod
    def factory_method(self, id):
        pass
    @property
    def client_factory_name(self):
        return self._CLIENT_FACTORY_NAME
class SportShopClientFactory(ClientFactory):
    _CLIENT_FACTORY_NAME = 'SportShopClientFactory'
    def factory_method(self, id) -> Clients:
        print('{}: Create new client with id = {}'.format(self.  CLIENT  FACTORY  NAME, id))
        return SportShopClient(id)
class ElectronicsShopClientFactory(ClientFactory):
    _CLIENT_FACTORY_NAME = 'ElectronicsShopClientFactory'
    def factory_method(self, id) -> Clients:
        print('{}: Create new client with id = {}'.format(self._CLIENT_FACTORY_NAME, id))
        return ElectronicsShopClient(id)
class SportElectronicsShopClientFactory(ClientFactory):
    _CLIENT_FACTORY_NAME = 'SportElectronicsShopClientFactory'
    def factory_method(self, id) -> Clients:
        print('{}: Create new client with id = {}'.format(self._CLIENT_FACTORY_NAME, id))
        return SportElectronicsShopClient(id)
def get  client(factory: ClientFactory, id):
    return factory.factory_method(id)
```

Facade.py

```
from __future__ import annotations
from lab_python_pt.factory.ShopFactory import SportShopFactory, ElectronicsShopFactory, get_shop
from lab_python_pt.factory.ClientFactory import SportShopClientFactory, ElectronicsShopClientFactory, \
    SportElectronicsShopClientFactory, get_client
class Facade:
    def __init__(self, sport_shops,
                 electronics_shops,
                 sport_shop_clients,
                 electronics  shop  clients,
                 sport_electronics_shop_clients):
        self.__sport_shops = sport_shops
        self.__electronics_shops = electronics_shops
        self.__sport_shop_clients = sport_shop_clients
        self.__electronics_shop_clients = electronics_shop_clients
        self.__sport_electronics_shop_clients = sport_electronics_shop_clients
    @property
    def sport_shops(self):
        return self.__sport_shops
    @property
    def electronics_shops(self):
```

```
    return self.__electronics_shops
@property
def sport_shop_clients(self):
    return self.__sport_shop_clients
@property
def electronics_shop_clients(self):
    return self.__electronics_shop_clients
@property
def sport_electronics_shop_clients(self):
    return self.__sport_electronics_shop_clients
def sport_shop_business_logic(self):
    print('Sport shop business logic:')
    for i in range(0, len(self.__sport_shops)):
        self.__sport_shops[i].business_logic()
    print('\n')
def electronic_shop_business_logic(self):
    print('Electronics shop business logic:')
    for i in range(0, len(self.    electronics  shops)):
        self.__electronics_shops[i].business_logic()
    print('\n')
def create_shops(self, sport_shop_count, electronics_shops_count):
    print('Factory shops:')
    self.__create_shops('sport',
                        self.    sport  shops,
                        sport_shop_count,
                        SportShopFactory())
    self.__create_shops('electronics',
                        self.__electronics_shops,
                        electronics_shops_count,
                        ElectronicsShopFactory())
    print('\n')
def create_clients(self,
                   sport_shop_clients_count,
                   electronics_shop_clients_count,
                   sport_electronics_shop_clients_count):
    print('\nFactory clients:')
    self.__create_clients('sport',
                          self.__sport_shop_clients,
                          sport_shop_clients_count,
                          SportShopClientFactory())
    self.__create_clients('electronics',
                          self.__electronics_shop_clients,
                          electronics_shop_clients_count,
                          ElectronicsShopClientFactory())
    self.    create  clients('sport electronics',
                          self.__sport_electronics_shop_clients,
                          sport_electronics_shop_clients_count,
                          SportElectronicsShopClientFactory())
    print('\n')
def attach_clients(self):
    print('Observer attach:')
    self.__attach_clients('sport',
                          self.__sport_shops,
                          self.__sport_shop_clients)
    self.__attach_clients('electronics',
                          self.__electronics_shops,
                          self.    electronics  shop  clients)
    self.__attach_clients('sport electronics',
                          self.    sport  shops,
                          self.__sport_electronics_shop_clients)
    self.__attach_clients('sport electronics',
                          self.__electronics_shops,
                          self.__sport_electronics_shop_clients)
    print('\n')
def detach_clients(self):
    print('Observer detach:')
    self.__detach_clients('sport',
                          self.__sport_shops,
                          self.__sport_shop_clients)
    self.__detach_clients('electronics',
```

```
                    self.__electronics_shops,
                    self.__electronics_shop_clients)
        self.__detach_clients('sport electronics',
                    self.__sport_shops,
                    self.__sport_electronics_shop_clients)
        self.__detach_clients('sport electronics',
                    self.__electronics_shops,
                    self.__sport_electronics_shop_clients)
        print('\n')
    def __create_shops(self, str, shops_list, count, factory):
        print('\nCreate {} {} shops:'.format(count, str))
        for i in range(0, count):
            shops_list.append(get_shop(factory, i))
    def __create_clients(self, str, clients_list, count, factory):
        print('\nCreate {} {} shop clients:'.format(count, str))
        for i in range(0, count):
            clients_list.append(get_client(factory, i))
    def    attach_clients(self, str, shop_list, clients_list):
        print('\nAttach {} {} shop clients:'.format(len(clients_list), str))
        for i in range(0, len(shop_list)):
            for j in range(0, len(clients_list)):
                shop_list[i].attach(clients_list[j])
    def __detach_clients(self, str, shop_list, clients_list):
        print('\nDetach {} {} shop clients:'.format(len(clients_list), str))
        for i in range(0, len(shop_list)):
            for j in range(0, len(clients_list)):
                shop_list[i].detach(clients_list[j])
```

Текст тестов

tests_observer.py

```
import unittest
from lab_python_pt.observer.Observer import \
    SportShop, \
    SportShopClient, \
    ElectronicsShop, \
    ElectronicsShopClient, \
    SportElectronicsShopClient
n = 10
observer_version = [[SportShop, SportShopClient, 5],
                    [ElectronicsShop, ElectronicsShopClient, 7],
                    [SportShop, SportElectronicsShopClient, 5],
                    [ElectronicsShop, SportElectronicsShopClient, 7]]
class TestObserver(unittest.TestCase):
    def test_observers(self):
        for j in range(1, 20):
            for obs in observer_version:
                with self.subTest(j=j, obs=obs):
                    shop = obs[0](0, j)
                    shop_clients = []
                    for i in range(0, n):
                        shop_clients.append(obs[1](i))
                    for i in range(0, n):
                        shop.attach(shop_clients[i])
                    shop.business_logic()
                    for i in range(0, n):
                        if j < obs[2]:
                            self.assertEqual(shop_clients[i].go_to_shop, False)
                        else:
                            self.assertEqual(shop_clients[i].go_to_shop, True)
                    for i in range(0, n):
                        shop.detach(shop_clients[i])
if __name__ == '__main__':
    unittest.main()
```

tests_shop_factory.py

```
import unittest
from lab_python_pt.factory.ShopFactory import \
    SportShopFactory, \
```

```
    ElectronicsShopFactory, \
    get_shop
factories = [[SportShopFactory(), 'SportShopFactory', 'SportShop'],
             [ElectronicsShopFactory(), 'ElectronicsShopFactory', 'ElectronicsShop']]
class TestsShopsFactory(unittest.TestCase):
    def test_create_factory(self):
        for j in range(1, 20):
            for factory in factories:
                with self.subTest(j=j, factory=factory):
                    self.assertEqual(factory[0].shop_factory_name, factory[1])
                    client = get_shop(factory[0], j)
                    self.assertEqual(client.id, j)
                    self.assertEqual(client.get_shop_name(), factory[2])
if __name__ == '__main__':
    unittest.main()
```

tests_client_factory.py

```
import unittest
from lab_python_pt.factory.ClientFactory import \
    SportShopClientFactory, \
    ElectronicsShopClientFactory, \
    SportElectronicsShopClientFactory, \
    get_client
factories = [[SportShopClientFactory(), 'SportShopClientFactory', 'SportShopClient'],
             [ElectronicsShopClientFactory(), 'ElectronicsShopClientFactory', 'ElectronicsShopClient'],
             [SportElectronicsShopClientFactory(), 'SportElectronicsShopClientFactory', 'SportElectronicsShopClient']]
class TestsClientsFactory(unittest.TestCase):
    def test_create_factory(self):
        for j in range(1, 20):
            for factory in factories:
                with self.subTest(j=j, factory=factory):
                    self.assertEqual(factory[0].client_factory_name, factory[1])
                    client = get_client(factory[0], j)
                    self.assertEqual(client.id, j)
                    self.assertEqual(client.get_client_name(), factory[2])
if __name__ == '__main__':
    unittest.main()
```

tests_facade.py

```
import unittest
from lab_python_pt.facade.Facade import Facade
shops = ['SportShop', 'ElectronicsShop']
clients = ['SportShopClient', 'ElectronicsShopClient', 'SportElectronicsShopClient']
class TestsFacade(unittest.TestCase):
    def test_facade_create_shop(self):
        for i in range(1, 20):
            for j in range(1, 20):
                with self.subTest(i=i, j=j):
                    facade = Facade()
                    facade.create_shops(i, j)
                    sport_shops = facade.sport_shops
                    electronics_shops = facade.electronics_shops
                    for sp in sport_shops:
                        self.assertEqual(sp.get_shop_name(), shops[0])
                    for es in electronics_shops:
                        self.assertEqual(es.get_shop_name(), shops[1])
    def test_facade_create_clients(self):
        for i in range(1, 20):
            for j in range(1, 20):
                for k in range(1, 20):
                    with self.subTest(i=i, j=j, k=k):
                        facade = Facade()
                        facade.create_clients(i, j, k)
                        sport_shops_clients = facade.sport_shop_clients
                        electronics_shops_clients = facade.electronics_shop_clients
                        sport_electronics_shop_clients = facade.sport_electronics_shop_clients
                        for ssc in sport_shops_clients:
                            self.assertEqual(ssc.get  client  name(), clients[0])
                        for esc in electronics_shops_clients:
```

```
                self.assertEqual(esc.get_client_name(), clients[1])
            for sesc in sport_electronics_shop_clients:
                self.assertEqual(sesc.get_client_name(), clients[2])
if __name__ == '__main__':
    unittest.main()
```

main_tests.py

```
import unittest
from lab_python_pt.observer.tests.tests_observer import TestObserver
from lab_python_pt.factory.tests.tests_client_factory import TestsClientsFactory
from lab_python_pt.factory.tests.tests_shop_factory import TestsShopsFactory
if __name__ == '__main__':
    unittest.main()
```

Примеры работы программы

```
Factory shops:
Create 1 sport shops:
SportShopFactory: Create new shop with id = 0
Create 2 electronics shops:
ElectronicsShopFactory: Create new shop with id = 0
ElectronicsShopFactory: Create new shop with id = 1
Factory clients:
Create 1 sport shop clients:
SportShopClientFactory: Create new client with id = 0
Create 2 electronics shop clients:
ElectronicsShopClientFactory: Create new client with id = 0
ElectronicsShopClientFactory: Create new client with id = 1
Create 1 sport electronics shop clients:
SportElectronicsShopClientFactory: Create new client with id = 0
Observer attach:
Attach 1 sport shop clients:
SportShop 0: Attached an observer = SportShopClient 0
Attach 2 electronics shop clients:
ElectronicsShop 0: Attached an observer = ElectronicsShopClient 0
ElectronicsShop 0: Attached an observer = ElectronicsShopClient 1
ElectronicsShop 1: Attached an observer = ElectronicsShopClient 0
ElectronicsShop 1: Attached an observer = ElectronicsShopClient 1
Attach 1 sport electronics shop clients:
SportShop 0: Attached an observer = SportElectronicsShopClient 0
Attach 1 sport electronics shop clients:
ElectronicsShop 0: Attached an observer = SportElectronicsShopClient 0
ElectronicsShop 1: Attached an observer = SportElectronicsShopClient 0
Sport shop business logic:
SportShop 0: I received 4 new items
SportShop 0: 2 observers
SportShop 0: Notifying observers...
Electronics shop business logic:
ElectronicsShop 0: I received 12 new items
ElectronicsShop 0: 3 observers
ElectronicsShop 0: Notifying observers...
ElectronicsShopClient 0: Reacted to the event
ElectronicsShopClient 1: Reacted to the event
SportElectronicsShopClient 0: Reacted to the event
ElectronicsShop 1: I received 2 new items
ElectronicsShop 1: 3 observers
ElectronicsShop 1: Notifying observers...
Observer detach:
Detach 1 sport shop clients:
SportShop 0: Detached an observer = SportShopClient 0
Detach 2 electronics shop clients:
ElectronicsShop 0: Detached an observer = ElectronicsShopClient 0
ElectronicsShop 0: Detached an observer = ElectronicsShopClient 1
ElectronicsShop 1: Detached an observer = ElectronicsShopClient 0
ElectronicsShop 1: Detached an observer = ElectronicsShopClient 1
Detach 1 sport electronics shop clients:
SportShop 0: Detached an observer = SportElectronicsShopClient 0
Detach 1 sport electronics shop clients:
ElectronicsShop 0: Detached an observer = SportElectronicsShopClient 0
```

ElectronicsShop 1: Detached an observer = SportElectronicsShopClient 0
Sport shop business logic:
SportShop 0: I received 4 new items
SportShop 0: 0 observers
Electronics shop business logic:
ElectronicsShop 0: I received 12 new items
ElectronicsShop 0: 0 observers
ElectronicsShop 1: I received 2 new items
ElectronicsShop 1: 0 observers

Примеры работы тестов

..

Ran 5 tests in 1.579s

OK