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

**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №4

Выполнил:

студент группы ИУ5Ц-72Б Гусев С.Р.

Преподаватель:

Гапанюк Ю.Е

2020

1. Описание задания
2. 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/facade)
3. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
4. Текст программы
5. **Фасад шаблон проектирования (Facade.py)**

В качестве порождающего шаблона используется паттерн фасад -это структурный паттерн проектирования, который предоставляет простой интерфейс к сложной системе классов, библиотеке или фреймворку. В данной задаче приведен пример применения этого паттерна к предметной области «Магазин».

Текст программы:

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:

""" В зависимости от потребностей вашего приложения вы можете предоставить

Фасаду существующие объекты подсистемы или заставить Фасад создать их

самостоятельно.

"""

def \_\_init\_\_(self, sport\_shops=None,

electronics\_shops=None,

sport\_shop\_clients=None,

electronics\_shop\_clients=None,

sport\_electronics\_shop\_clients=None):

if sport\_electronics\_shop\_clients is None:

sport\_electronics\_shop\_clients = []

if electronics\_shop\_clients is None:

electronics\_shop\_clients = []

if sport\_shop\_clients is None:

sport\_shop\_clients = []

if electronics\_shops is None:

electronics\_shops = []

if sport\_shops is None:

sport\_shops = []

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('Бизнес-логика спортивного магазина:')

for i in range(0, len(self.\_\_sport\_shops)):

self.\_\_sport\_shops[i].business\_logic()

print('\n')

def electronic\_shop\_business\_logic(self):

print('Бизнес-логика магазина электроники:')

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('Заводские магазины:')

self.\_\_create\_shops('спорт',

self.\_\_sport\_shops,

sport\_shop\_count,

SportShopFactory())

self.\_\_create\_shops('электроника',

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('\nЗаводской клиент:')

self.\_\_create\_clients('спорт',

self.\_\_sport\_shop\_clients,

sport\_shop\_clients\_count,

SportShopClientFactory())

self.\_\_create\_clients('элетроника',

self.\_\_electronics\_shop\_clients,

electronics\_shop\_clients\_count,

ElectronicsShopClientFactory())

self.\_\_create\_clients('спорт электроника',

self.\_\_sport\_electronics\_shop\_clients,

sport\_electronics\_shop\_clients\_count,

SportElectronicsShopClientFactory())

print('\n')

def attach\_clients(self):

print('Прикрепление наблюдателя:')

self.\_\_attach\_clients('спорт',

self.\_\_sport\_shops,

self.\_\_sport\_shop\_clients)

self.\_\_attach\_clients('электроника',

self.\_\_electronics\_shops,

self.\_\_electronics\_shop\_clients)

self.\_\_attach\_clients('спорт электроника',

self.\_\_sport\_shops,

self.\_\_sport\_electronics\_shop\_clients)

self.\_\_attach\_clients('спорт электроника',

self.\_\_electronics\_shops,

self.\_\_sport\_electronics\_shop\_clients)

print('\n')

"""

Методы Фасада удобны для быстрого доступа к сложной функциональности

подсистем. Однако клиенты получают только часть возможностей подсистемы.

"""

def detach\_clients(self):

print('Прикрепление наблюдателя:')

self.\_\_detach\_clients('спорт',

self.\_\_sport\_shops,

self.\_\_sport\_shop\_clients)

self.\_\_detach\_clients('электроника',

self.\_\_electronics\_shops,

self.\_\_electronics\_shop\_clients)

self.\_\_detach\_clients('электроника спорт',

self.\_\_sport\_shops,

self.\_\_sport\_electronics\_shop\_clients)

self.\_\_detach\_clients('спорт электроника',

self.\_\_electronics\_shops,

self.\_\_sport\_electronics\_shop\_clients)

print('\n')

def \_\_create\_shops(self, str, shops\_list, count, factory):

print('\nСоздать {} {} магазины:'.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('\nСоздать {} {} магазин клиенты:'.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('\nПрикрепить {} {} магазин клиенты:'.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):

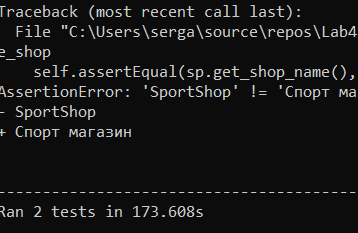
print('\nОткрепить {} {} магазин клиенты:'.format(len(clients\_list), str))

for i in range(0, len(shop\_list)):

for j in range(0, len(clients\_list)):

shop\_list[i].detach(clients\_list[j])

Экранные формы:



1. **ClientFactory.py and ShopFactory.py**

Текст программы:

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('{}: Создать нового клиента с id = {}'.format(self.\_CLIENT\_FACTORY\_NAME, id))

return SportShopClient(id)

class ElectronicsShopClientFactory(ClientFactory):

\_CLIENT\_FACTORY\_NAME = 'ElectronicsShopClientFactory'

def factory\_method(self, id) -> Clients:

print('{}: Создать нового клиента с id = {}'.format(self.\_CLIENT\_FACTORY\_NAME, id))

return ElectronicsShopClient(id)

class SportElectronicsShopClientFactory(ClientFactory):

\_CLIENT\_FACTORY\_NAME = 'SportElectronicsShopClientFactory'

def factory\_method(self, id) -> Clients:

print('{}: Создать нового клиента с id = {}'.format(self.\_CLIENT\_FACTORY\_NAME, id))

return SportElectronicsShopClient(id)

def get\_client(factory: ClientFactory, id):

return factory.factory\_method(id)

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('{}: Создать нового клиента с id = {}'.format(self.\_SHOP\_FACTORY\_NAME, id))

return SportShop(id)

class ElectronicsShopFactory(ShopFactory):

\_SHOP\_FACTORY\_NAME = 'ElectronicsShopFactory'

def factory\_method(self, id) -> Shops:

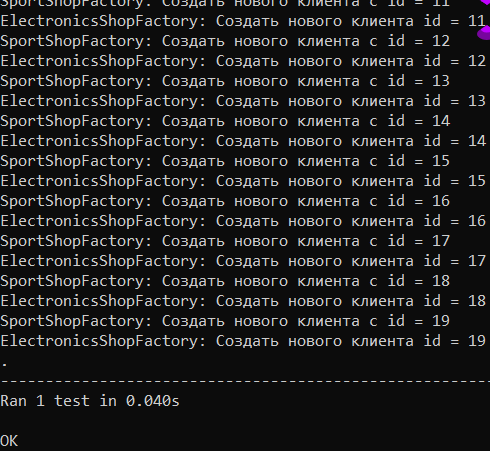
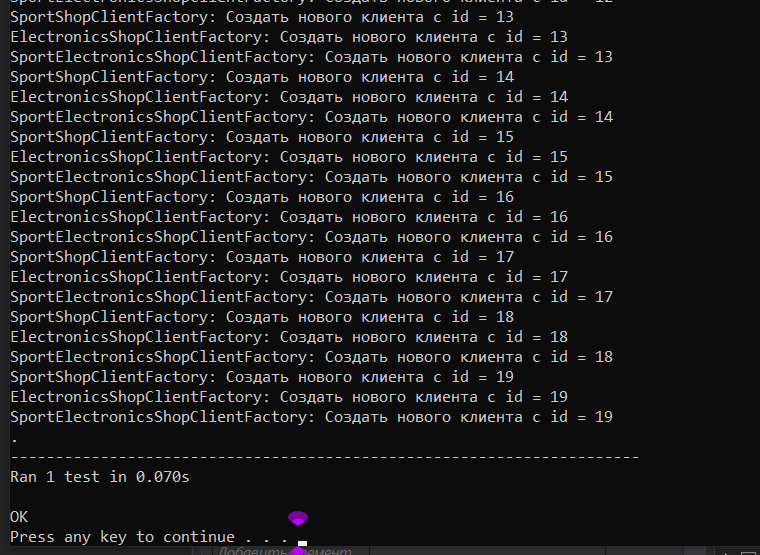
print('{}: Создать нового клиента id = {}'.format(self.\_SHOP\_FACTORY\_NAME, id))

return ElectronicsShop(id)

def get\_shop(factory: ShopFactory, id):

return factory.factory\_method(id)

Экранные формы:



1. **Observe.py**

Текст программы:

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

Экранные формы:

