**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет "Информатика и системы управления"

Кафедра ИУ5 "Системы обработки информации и управления"

Разработка интернет-приложений

Отчет по лабораторной работе №4

Выполнила: Пасатюк Александра, группа ИУ5-53Б

Москва, 2020

**Задание**

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

* TDD – фреймворк
* BDD – фреймворк
* Создание Mock-объектов

**Текст программ**

**Порождающий паттерн (абстрактная фабрика)**

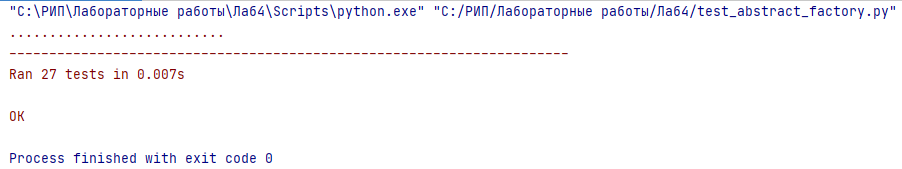
**abstract\_factory.py**

**from** abc **import** ABC, abstractmethod  
  
  
**def** define\_resolution(platform):  
 **if** platform == **"Стационарный ПК"**:  
 **return "2560x1600"  
 elif** platform == **"Переносной ПК"**:  
 **return "1600x1200"  
 elif** platform == **"Смартфон"**:  
 **return "1280x1024"***# абстрактный класс окна с абстрактным методом его отрисовки***class** Window(ABC):  
  
 @abstractmethod  
 **def** paint(self, resolution):  
 **pass***# абстрактный класс кнопки с абстрактным методом ее отрисовки***class** Button(ABC):  
  
 @abstractmethod  
 **def** paint(self):  
 **pass***# абстрактный класс чек-бокса с абстрактным методом его отрисовки***class** Checkbox(ABC):  
  
 @abstractmethod  
 **def** paint(self):  
 **pass** @abstractmethod  
 **def** paint\_with\_button(self, button):  
 **pass***# абстрактный класс текстового поля с абстрактным методом его отрисовки***class** Textfield(ABC):  
  
 @abstractmethod  
 **def** paint(self):  
 **pass***# Абстрактная фабрика***class** GUIFactory(ABC):  
  
 @abstractmethod  
 **def** create\_window(self):  
 **pass** @abstractmethod  
 **def** create\_button(self):  
 **pass** @abstractmethod  
 **def** create\_checkbox(self):  
 **pass** @abstractmethod  
 **def** create\_textfield(self):  
 **pass***# класс окна для Windows***class** WinWindow(Window):  
  
 **def** paint(self, resolution):  
 **return f"Создание и отрисовка окна с разрешением {**resolution**} в стиле Windows"***# класс окна для macOS***class** MacWindow(Window):  
  
 **def** paint(self, resolution):  
 **return f"Создание и отрисовка окна с разрешением {**resolution**} в стиле macOS"***# класс окна для Linux***class** LinuxWindow(Window):  
  
 **def** paint(self, resolution):  
 **return f"Создание и отрисовка окна с разрешением {**resolution**} в стиле Linux"***# класс кнопки для Windows***class** WinButton(Button):  
  
 **def** paint(self):  
 **return "Отрисовка кнопки в стиле Windows"***# класс кнопки для macOS***class** MacButton(Button):  
  
 **def** paint(self):  
 **return "Отрисовка кнопки в стиле macOS"***# класс кнопки для Linux***class** LinuxButton(Button):  
  
 **def** paint(self):  
 **return "Отрисовка кнопки в стиле Linux"***# класс чек-бокса для Windows***class** WinCheckbox(Checkbox):  
  
 **def** paint(self):  
 **return "Отрисовка чек-бокса в стиле Windows"  
  
 def** paint\_with\_button(self, button):  
 **if** type(button) == WinButton:  
 result = button.paint()  
 **return f"Отрисовка чек-бокса и {**result**}"  
 else**:  
 **raise** ValueError  
  
  
*# класс чек-бокса для macOS***class** MacCheckbox(Checkbox):  
  
 **def** paint(self):  
 **return "Отрисовка чек-бокса в стиле macOS"  
  
 def** paint\_with\_button(self, button):  
 **if** type(button) == MacButton:  
 result = button.paint()  
 **return f"Отрисовка чек-бокса и {**result**}"  
 else**:  
 **raise** ValueError  
  
  
*# класс чек-бокса для Linux***class** LinuxCheckbox(Checkbox):  
  
 **def** paint(self):  
 **return "Отрисовка чек-бокса в стиле Linux"  
  
 def** paint\_with\_button(self, button):  
 **if** type(button) == LinuxButton:  
 result = button.paint()  
 **return f"Отрисовка чек-бокса и {**result**}"  
 else**:  
 **raise** ValueError  
  
  
*# класс текстового поля для Windows***class** WinTextfield(Textfield):  
  
 **def** paint(self):  
 **return "Отрисовка текстового поля в стиле Windows"***# класс текстового поля для macOS***class** MacTextfield(Textfield):  
  
 **def** paint(self):  
 **return "Отрисовка текстового поля в стиле macOS"***# класс текстового поля для Linux***class** LinuxTextfield(Textfield):  
  
 **def** paint(self):  
 **return "Отрисовка текстового поля в стиле Linux"***# фабрика для Windows***class** WinFactory(GUIFactory):  
  
 **def** create\_window(self):  
 **return** WinWindow()  
  
 **def** create\_button(self):  
 **return** WinButton()  
  
 **def** create\_checkbox(self):  
 **return** WinCheckbox()  
  
 **def** create\_textfield(self):  
 **return** WinTextfield()  
  
  
*# фабрика для macOS***class** MacFactory(GUIFactory):  
  
 **def** create\_window(self):  
 **return** MacWindow()  
  
 **def** create\_button(self):  
 **return** MacButton()  
  
 **def** create\_checkbox(self):  
 **return** MacCheckbox()  
  
 **def** create\_textfield(self):  
 **return** MacTextfield()  
  
  
*# фабрика для Linux***class** LinuxFactory(GUIFactory):  
  
 **def** create\_window(self):  
 **return** LinuxWindow()  
  
 **def** create\_button(self):  
 **return** LinuxButton()  
  
 **def** create\_checkbox(self):  
 **return** LinuxCheckbox()  
  
 **def** create\_textfield(self):  
 **return** LinuxTextfield()  
  
  
*# клиентский код***def** client\_code(factory):  
 window = factory.create\_window()  
 button = factory.create\_button()  
 checkbox = factory.create\_checkbox()  
 textfield = factory.create\_textfield()  
  
 print(window.paint(define\_resolution(**"Стационарный ПК"**)))  
 print(window.paint(define\_resolution(**"Переносной ПК"**)))  
 print(window.paint(define\_resolution(**"Смартфон"**)))  
 print(button.paint())  
 print(checkbox.paint())  
 print(textfield.paint())  
 print(checkbox.paint\_with\_button(button))  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 print(**"Клиентский код на Windows"**)  
 client\_code(WinFactory())  
  
 print(**'\n'**)  
  
 print(**"Клиентский код на macOS"**)  
 client\_code(MacFactory())  
  
 print(**'\n'**)  
  
 print(**"Клиентский код на Linux"**)  
 client\_code(LinuxFactory())

**test\_abstract\_factory.py**

**import** unittest  
**from** unittest **import** TestCase  
**from** unittest.mock **import** patch  
**from** abstract\_factory **import** WinFactory  
**from** abstract\_factory **import** MacFactory  
**from** abstract\_factory **import** LinuxFactory  
  
  
**class** AbstractFactoryTestCase(TestCase):  
 **def** start(self):  
 factory = WinFactory()  
  
 *# проверка верной отрисовки окна на Windows с разрешением 2560x1600  
 # функцию define\_resolution делаем Mock-объектом,  
 # т.к. нам важно проверить, чтобы правильно отрисовывалось окно при определенном разрешении,  
 # а не логику функции нахождения разрешения* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"2560x1600"**)  
 **def** test\_win\_window\_hr(self, define\_resolution):  
 factory = WinFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 2560x1600 в стиле Windows"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на Windows с разрешением 1600x1200  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлому тесту* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1600x1200"**)  
 **def** test\_win\_window\_mr(self, define\_resolution):  
 factory = WinFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1600x1200 в стиле Windows"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на Windows с разрешением 1280x1024  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлым тестам* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1280x1024"**)  
 **def** test\_win\_window\_lr(self, define\_resolution):  
 factory = WinFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1280x1024 в стиле Windows"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки кнопки на Windows* **def** test\_win\_button(self):  
 factory = WinFactory()  
 button = factory.create\_button()  
 self.assertEqual(**"Отрисовка кнопки в стиле Windows"**, button.paint())  
  
 *# проверка верной отрисовки чек-бокса на Windows* **def** test\_win\_checkbox(self):  
 factory = WinFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса в стиле Windows"**, checkbox.paint())  
  
 *# проверка верной отрисовки чек-бокса с кнопкой на Windows* **def** test\_win\_checkbox\_button(self):  
 factory = WinFactory()  
 button = factory.create\_button()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса и Отрисовка кнопки в стиле Windows"**, checkbox.paint\_with\_button(button))  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а textfield* **def** test\_win\_error\_textfield\_checkbox\_button(self):  
 factory = WinFactory()  
 textfield = factory.create\_textfield()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, textfield)  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а checkbox* **def** test\_win\_error\_checkbox\_checkbox\_button(self):  
 factory = WinFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, checkbox)  
  
 *# проверка верной отрисовки текстового поля на Windows* **def** test\_win\_textfield(self):  
 factory = WinFactory()  
 textfield = factory.create\_textfield()  
 self.assertEqual(**"Отрисовка текстового поля в стиле Windows"**, textfield.paint())  
  
 *# проверка верной отрисовки окна на macOS с разрешением 2560x1600  
 # функцию define\_resolution делаем Mock-объектом,  
 # т.к. нам важно проверить, чтобы правильно отрисовывалось окно при определенном разрешении,  
 # а не логику функции нахождения разрешения* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"2560x1600"**)  
 **def** test\_mac\_window\_hr(self, define\_resolution):  
 factory = MacFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 2560x1600 в стиле macOS"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на macOS с разрешением 1600x1200  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлому тесту* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1600x1200"**)  
 **def** test\_mac\_window\_mr(self, define\_resolution):  
 factory = MacFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1600x1200 в стиле macOS"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на macOS с разрешением 1280x1024  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлым тестам* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1280x1024"**)  
 **def** test\_mac\_window\_lr(self, define\_resolution):  
 factory = MacFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1280x1024 в стиле macOS"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки кнопки на macOS* **def** test\_mac\_button(self):  
 factory = MacFactory()  
 button = factory.create\_button()  
 self.assertEqual(**"Отрисовка кнопки в стиле macOS"**, button.paint())  
  
 *# проверка верной отрисовки чек-бокса на macOS* **def** test\_mac\_checkbox(self):  
 factory = MacFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса в стиле macOS"**, checkbox.paint())  
  
 *# проверка верной отрисовки чек-бокса с кнопкой на macOS* **def** test\_mac\_checkbox\_button(self):  
 factory = MacFactory()  
 button = factory.create\_button()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса и Отрисовка кнопки в стиле macOS"**, checkbox.paint\_with\_button(button))  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а textfield* **def** test\_mac\_error\_textfield\_checkbox\_button(self):  
 factory = MacFactory()  
 textfield = factory.create\_textfield()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, textfield)  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а checkbox* **def** test\_mac\_error\_checkbox\_checkbox\_button(self):  
 factory = MacFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, checkbox)  
  
 *# проверка верной отрисовки текстового поля на macOS* **def** test\_mac\_textfield(self):  
 factory = MacFactory()  
 textfield = factory.create\_textfield()  
 self.assertEqual(**"Отрисовка текстового поля в стиле macOS"**, textfield.paint())  
  
 *# проверка верной отрисовки окна на Linux с разрешением 2560x1600  
 # функцию define\_resolution делаем Mock-объектом,  
 # т.к. нам важно проверить, чтобы правильно отрисовывалось окно при определенном разрешении,  
 # а не логику функции нахождения разрешения* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"2560x1600"**)  
 **def** test\_linux\_window\_hr(self, define\_resolution):  
 factory = LinuxFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 2560x1600 в стиле Linux"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на Linux с разрешением 1600x1200  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлому тесту* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1600x1200"**)  
 **def** test\_linux\_window\_mr(self, define\_resolution):  
 factory = LinuxFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1600x1200 в стиле Linux"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки окна на Linux с разрешением 1280x1024  
 # функцию define\_resolution делаем Mock-объектом аналогично прошлым тестам* @patch(**'abstract\_factory.define\_resolution'**, return\_value=**"1280x1024"**)  
 **def** test\_linux\_window\_lr(self, define\_resolution):  
 factory = LinuxFactory()  
 window = factory.create\_window()  
 self.assertEqual(**"Создание и отрисовка окна с разрешением 1280x1024 в стиле Linux"**,  
 window.paint(define\_resolution(**"platform"**)))  
  
 *# проверка верной отрисовки кнопки на Linux* **def** test\_linux\_button(self):  
 factory = LinuxFactory()  
 button = factory.create\_button()  
 self.assertEqual(**"Отрисовка кнопки в стиле Linux"**, button.paint())  
  
 *# проверка верной отрисовки чек-бокса на Linux* **def** test\_linux\_checkbox(self):  
 factory = LinuxFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса в стиле Linux"**, checkbox.paint())  
  
 *# проверка верной отрисовки чек-бокса с кнопкой на Linux* **def** test\_linux\_checkbox\_button(self):  
 factory = LinuxFactory()  
 button = factory.create\_button()  
 checkbox = factory.create\_checkbox()  
 self.assertEqual(**"Отрисовка чек-бокса и Отрисовка кнопки в стиле Linux"**, checkbox.paint\_with\_button(button))  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а textfield* **def** test\_linux\_error\_textfield\_checkbox\_button(self):  
 factory = LinuxFactory()  
 textfield = factory.create\_textfield()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, textfield)  
  
 *# проверка на исключение, в случае передачи в checkbox.paint\_with\_button() не button, а checkbox* **def** test\_linux\_error\_checkbox\_checkbox\_button(self):  
 factory = LinuxFactory()  
 checkbox = factory.create\_checkbox()  
 self.assertRaises(ValueError, checkbox.paint\_with\_button, checkbox)  
  
 *# проверка верной отрисовки текстового поля на Linux* **def** test\_linux\_textfield(self):  
 factory = LinuxFactory()  
 textfield = factory.create\_textfield()  
 self.assertEqual(**"Отрисовка текстового поля в стиле Linux"**, textfield.paint())  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

**Результат выполнения теста**



**Поведенческий паттерн (наблюдатель)**

**Observer.py**

**from** abc **import** ABC, abstractmethod  
  
  
*# абстрактный класс издателя***class** Publisher(ABC):  
  
 @abstractmethod  
 **def** attach(self, subscriber):  
 **pass** @abstractmethod  
 **def** detach(self, subscriber):  
 **pass** @abstractmethod  
 **def** notify(self):  
 **pass***# абстрактный класс подписчика(наблюдателя)***class** Subscriber(ABC):  
  
 @abstractmethod  
 **def** update(self, publisher):  
 **pass***# магазин, оповещающий подписчиков***class** StorePublisher(Publisher):  
  
 **def** \_\_init\_\_(self):  
 self.new\_goods = **''** self.subscribers = []  
  
 **def** attach(self, subscriber):  
 self.subscribers.append(subscriber)  
 **return** (**"Publisher:"** + **f"Добавлен новый подписчик с ником {**subscriber.name**}"**)  
  
 **def** detach(self, subscriber):  
 self.subscribers.remove(subscriber)  
 **return** (**"Publisher:"** + **f"Удален подписчик с ником {**subscriber.name**}"**)  
  
 **def** notify(self):  
 print(**"Publisher:"**, **'red'**), **"Оповещаю подписчиков..."** subscribers\_reacts = []  
 **for** subscriber **in** self.subscribers:  
 subscribers\_reacts.append(subscriber.update(self))  
 **for** react **in** subscribers\_reacts:  
 **if** react != 1:  
 print(react)  
  
 **def** goods\_arrival(self, goods):  
 self.new\_goods = goods  
 print(**"Publisher:"**, **'red'**), **f"Поступил новый товар - {**self.new\_goods**}"** self.notify()  
  
  
*# Человек, подписавшиея на оповещения о поступлении кроссовок***class** SneakersSubscriber(Subscriber):  
  
 **def** \_\_init\_\_(self, name):  
 self.name = name  
  
 **def** update(self, publisher):  
 **if** publisher.new\_goods == **"кроссовки"**:  
 react = **"SneakersSubscriber:"** + **f"{**self.name**} реагирует на новое поступление кроссовок"  
 return** react  
 **else**:  
 **return** 1  
  
  
*# Человек, подписавшиеся на оповещения о поступлении толстовки***class** HoodiesSubscriber(Subscriber):  
  
 **def** \_\_init\_\_(self, name):  
 self.name = name  
  
 **def** update(self, publisher):  
 **if** publisher.new\_goods == **"толстовка"**:  
 react = **"SneakersSubscriber:"** + **f"{**self.name**} реагирует на новое поступление толстовки"  
 return** react  
 **else**:  
 **return** 1  
  
  
**def** client\_code():  
 store = StorePublisher()  
  
 first\_sneakers\_subscriber = SneakersSubscriber(**"Серега"**)  
 print(store.attach(first\_sneakers\_subscriber))  
 second\_sneakers\_subscriber = SneakersSubscriber(**"Сашка"**)  
 print(store.attach(second\_sneakers\_subscriber))  
 first\_hoodies\_subscriber = HoodiesSubscriber(**"Вика"**)  
 print(store.attach(first\_hoodies\_subscriber))  
  
 print(**'\n'**)  
  
 store.goods\_arrival(**"кроссовки"**)  
 store.goods\_arrival(**"толстовка"**)  
  
 print(**'\n'**)  
  
 print(store.detach(first\_sneakers\_subscriber))  
  
 print(**'\n'**)  
  
 store.goods\_arrival(**"кроссовки"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 client\_code()

**test\_observer.py**

**from** unittest **import** TestCase  
**from** observer **import** SneakersSubscriber  
**from** observer **import** HoodiesSubscriber  
**from** observer **import** StorePublisher  
  
  
**class** ObserverTestCase(TestCase):  
  
 *# проверка добавления нового подписчика* **def** test\_attach(self):  
 sneakers\_subscriber = SneakersSubscriber(**"Name1"**)  
 hoodies\_subscriber = HoodiesSubscriber(**"Name2"**)  
 store = StorePublisher()  
  
 store.attach(sneakers\_subscriber)  
 store.attach(hoodies\_subscriber)  
  
 self.assertEqual(type(sneakers\_subscriber), type(store.subscribers[0]))  
 self.assertEqual(type(hoodies\_subscriber), type(store.subscribers[1]))  
  
 *# проверка удаления подписчика* **def** test\_detach(self):  
 sneakers\_subscriber = SneakersSubscriber(**"Name1"**)  
 hoodies\_subscriber = HoodiesSubscriber(**"Name2"**)  
 store = StorePublisher()  
 store.attach(sneakers\_subscriber)  
 store.attach(hoodies\_subscriber)  
  
 store.detach(sneakers\_subscriber)  
  
 self.assertEqual(1, len(store.subscribers))  
 self.assertEqual(type(hoodies\_subscriber), type(store.subscribers[0]))  
  
 *# проверка реакции на поступление новых кроссовок людей, подписанных на кроссовки* **def** test\_react\_sneakers\_subscriber(self):  
 store = StorePublisher()  
 sneakers\_subscriber = SneakersSubscriber(**"Name1"**)  
 store.new\_goods = **"кроссовки"** self.assertEqual(**"SneakersSubscriber:"** +  
 **f"{**sneakers\_subscriber.name**} реагирует на новое поступление кроссовок"**,  
 sneakers\_subscriber.update(store))  
  
 *# проверка реакции на поступление новых кроссовок людей, не подписанных на кроссовки* **def** test\_noreact\_hoodies\_subscriber(self):  
 store = StorePublisher()  
 hoodies\_subscriber = HoodiesSubscriber(**"Name1"**)  
 store.new\_goods = **"кроссовки"** self.assertEqual(1, hoodies\_subscriber.update(store))  
  
 *# проверка реакции на поступление новых худи людей, подписанных на худи* **def** test\_react\_hoodies\_subscriber(self):  
 store = StorePublisher()  
 hoodies\_subscriber = HoodiesSubscriber(**"Name1"**)  
 store.new\_goods = **"толстовка"** self.assertEqual(**"SneakersSubscriber:"** +  
 **f"{**hoodies\_subscriber.name**} реагирует на новое поступление толстовки"**,  
 hoodies\_subscriber.update(store))  
  
 *# проверка реакции на поступление новых худи людей, не подписанных на худи* **def** test\_noreact\_sneakers\_subscriber(self):  
 store = StorePublisher()  
 sneakers\_subscriber = SneakersSubscriber(**"Name1"**)  
 store.new\_goods = **"толстовка"** self.assertEqual(1, sneakers\_subscriber.update(store))

**Результат выполнения теста**



**Структурный паттерн (адаптер)**

**adapter.py**

*# класс цилиндрических деталей***class** RoundDetail:  
  
 **def** \_\_init\_\_(self, radius):  
 self.radius = radius  
  
 **def** get\_radius(self):  
 **return** self.radius  
  
  
*# класс параллелипипедных деталей***class** SquareDetail:  
  
 **def** \_\_init\_\_(self, width):  
 self.width = width  
  
 *# для тестирования  
 # def get\_radius(self):  
 # return self.width* **def** get\_width(self):  
 **return** self.width  
  
  
*# класс круглых отверстий***class** RoundHole:  
  
 **def** \_\_init\_\_(self, radius):  
 self.radius = radius  
  
 **def** get\_radius(self):  
 **return** self.radius  
  
 **def** fits(self, round\_detail):  
  
 **if** self.get\_radius() == round\_detail.get\_radius():  
 **return f"Деталь подходит. "** \  
 **f"Радиус детали: {**round\_detail.get\_radius()**}, радиус отверстия {**self.get\_radius()**}"  
 else**:  
 **return f"Деталь не подходит. "** \  
 **f"Радиус детали: {**round\_detail.get\_radius()**}, радиус отверстия {**self.get\_radius()**}"***# адаптер***class** SquareDetailAdapter(RoundDetail):  
  
 **def** \_\_init\_\_(self, square\_detail):  
 self.square\_detail = square\_detail  
  
 **def** get\_radius(self):  
 **return** self.square\_detail.get\_width() / 2  
  
  
**def** client\_code():  
 hole = RoundHole(10)  
 round\_detail1 = RoundDetail(10)  
 round\_detail2 = RoundDetail(20)  
 square\_detail1 = SquareDetail(10)  
 square\_detail2 = SquareDetail(20)  
  
 print(**"Проверяем цилиндрические детали:"**)  
 print(hole.fits(round\_detail1))  
 print(hole.fits(round\_detail2))  
 *# не работает, т.к. параллелипипедная деталь не соответсвует круглому отверстию  
 # print(hole.fits(square\_detail1))* print(**'\n'**)  
  
 print(**"Проверяем параллелипипедные детали"**)  
 square\_detail\_adapter1 = SquareDetailAdapter(square\_detail1)  
 print(hole.fits(square\_detail\_adapter1))  
 square\_detail\_adapter2 = SquareDetailAdapter(square\_detail2)  
 print(hole.fits(square\_detail\_adapter2))  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 client\_code()

**steps.py**

**from** behave **import** \*  
**from** adapter **import** RoundDetail  
**from** adapter **import** RoundHole  
**from** adapter **import** SquareDetail  
**from** adapter **import** SquareDetailAdapter  
  
  
@given(**'size of round detail - radius "{detail\_size}" and size of round hole - "{hole\_radius}"'**)  
**def** step(context, detail\_size, hole\_radius):  
 context.round\_detail = RoundDetail(int(detail\_size))  
 context.hole = RoundHole(int(hole\_radius))  
  
  
@given(**'size of square detail - width "{detail\_size}" and size of round hole - "{hole\_radius}"'**)  
**def** step(context, detail\_size, hole\_radius):  
 context.square\_detail = SquareDetail(int(detail\_size))  
 context.hole = RoundHole(int(hole\_radius))  
  
  
@then(**'detail and hole compatible'**)  
**def** step(context):  
 **assert** context.hole.fits(context.round\_detail) == **f"Деталь подходит. "** \  
 **f"Радиус детали: {**context.round\_detail.get\_radius()**}, "** \  
 **f"радиус отверстия {**context.hole.get\_radius()**}"**, \  
 **"Тест не пройден"**@then(**'detail and hole incompatible'**)  
**def** step(context):  
 **assert** context.hole.fits(context.round\_detail) == **f"Деталь не подходит. "** \  
 **f"Радиус детали: {**context.round\_detail.get\_radius()**}, "** \  
 **f"радиус отверстия {**context.hole.get\_radius()**}"**, \  
 **"Тест не пройден"**@then(**'the square detail is not comparable to the round hole'**)  
**def** step(context):  
 f = 0  
 **try**:  
 context.hole.fits(context.square\_detail)  
 **except** AttributeError:  
 f = 1  
 **finally**:  
 **assert** f == 1, **"Тест не пройден"**@then(**'detail and hole compatible after conversion via wrapper'**)  
**def** step(context):  
 context.adapter = SquareDetailAdapter(context.square\_detail)  
 **assert** context.hole.fits(context.adapter) == **f"Деталь подходит. "** \  
 **f"Радиус детали: {**context.adapter.get\_radius()**}, "** \  
 **f"радиус отверстия {**context.hole.get\_radius()**}"**, \  
 **"Тест не пройден"**@then(**'detail and hole incompatible after conversion via wrapper'**)  
**def** step(context):  
 context.adapter = SquareDetailAdapter(context.square\_detail)  
 **assert** context.hole.fits(context.adapter) == **f"Деталь не подходит. "** \  
 **f"Радиус детали: {**context.adapter.get\_radius()**}, "** \  
 **f"радиус отверстия {**context.hole.get\_radius()**}"**, \  
 **"Тест не пройден"**

**tests\_main\_interface.feature**

**Feature**: Compatibility check  
  
 **Scenario**: Checking a round detail of suitable size  
 **Given** size of round detail - radius "**10**" and size of round hole - "**10**"  
 **Then** detail and hole compatible  
  
 **Scenario**: Checking a round detail of unsuitable size  
 **Given** size of round detail - radius "**20**" and size of round hole - "**10**"  
 **Then** detail and hole incompatible  
  
 **Scenario**: Checking a square detail  
 **Given** size of square detail - width "**10**" and size of round hole - "**10**"  
 **Then** the square detail is not comparable to the round hole

**tests\_main\_interface\_via\_adapter.feature**

**Feature**: Compatibility check via wrapper  
  
 **Scenario**: Checking a square detail of suitable size  
 **Given** size of square detail - width "**20**" and size of round hole - "**10**"  
 **Then** detail and hole compatible after conversion via wrapper  
  
 **Scenario**: Checking a square detail if unsuitable size  
 **Given** size of square detail - width "**10**" and size of round hole - "**10**"  
 **Then** detail and hole incompatible after conversion via wrapper

**Результат выполнения теста**

