Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»



**Отчет**

**Лабораторная работа № 4**

**По курсу «Разработка интернет приложений»**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-55Б

Лункин В.И.

"29" октября 2020 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

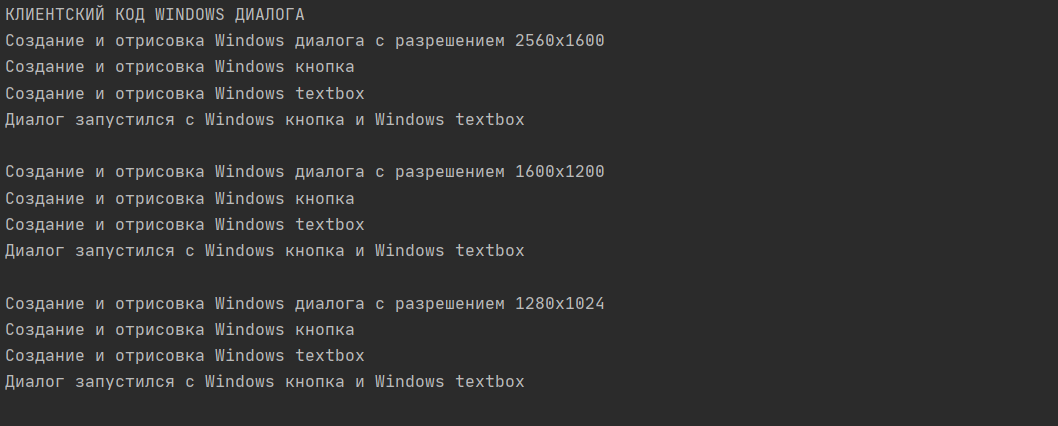
"\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

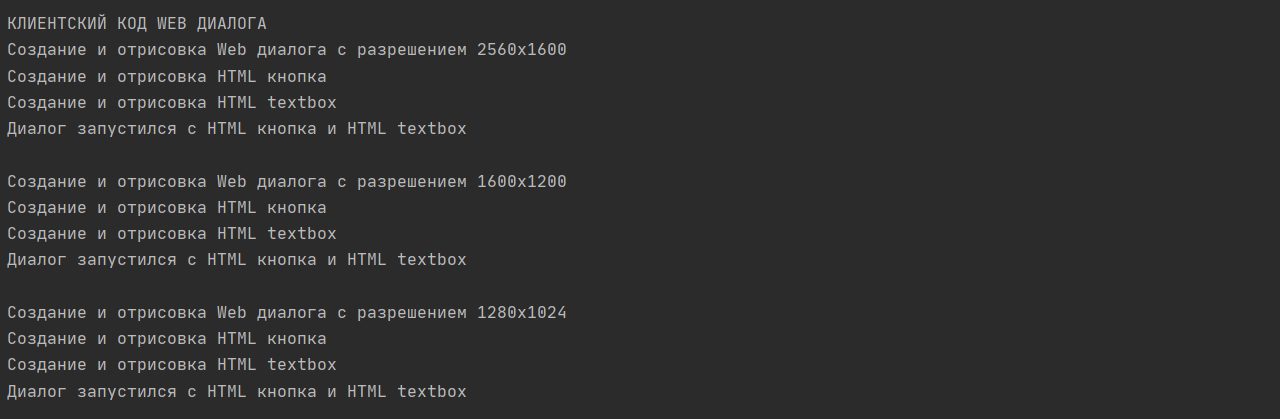
Москва 2020

1. **Общее 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:

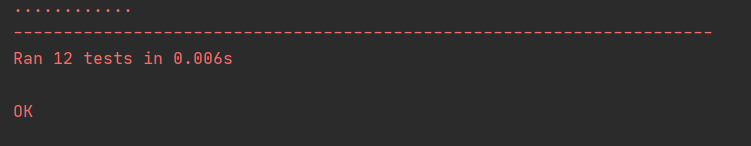
* TDD – фреймворк
* BDD – фреймворк
* Создание Mock-объектов

1. **Порождающий паттерн проектирования**
2. from \_\_future\_\_ import annotations  
   from abc import ABC, abstractmethod  
     
     
   def get\_res(platform):  
    if platform == "Настольный ПК":  
    return "2560x1600"  
    elif platform == "Ноутбук":  
    return "1600x1200"  
    elif platform == "Нетбук":  
    return "1280x1024"  
     
     
   class Dialog(ABC):  
    @abstractmethod  
    def create\_button(self):  
    pass  
     
    @abstractmethod  
    def create\_textbox(self):  
    pass  
     
    @abstractmethod  
    def paint(self, resolution):  
    pass  
     
    def render(self) -> str:  
    # Вызываем фабричный метод, чтобы получить объект-продукт.  
    button = self.create\_button()  
     
    textbox = self.create\_textbox()  
     
    # Далее, работаем с этим продуктом.  
    print(f"Создание и отрисовка {button.operation()}")  
    print(f"Создание и отрисовка {textbox.operation()}")  
    result = f"Диалог запустился с {button.operation()} и {textbox.operation()}"  
     
    return result  
     
     
   class WindowsDialog(Dialog):  
    def paint(self, resolution):  
    return f"Создание и отрисовка Windows диалога с разрешением {resolution}"  
     
    def create\_button(self) -> Button:  
    return WindowsButton()  
     
    def create\_textbox(self) -> Textbox:  
    return WindowsTextbox()  
     
     
   class WebDialog(Dialog):  
    def paint(self, resolution):  
    return f"Создание и отрисовка Web диалога с разрешением {resolution}"  
     
    def create\_button(self) -> Button:  
    return HTMLButton()  
     
    def create\_textbox(self) -> Textbox:  
    return HTMLTextbox()  
     
     
   class Button(ABC):  
    @abstractmethod  
    def operation(self) -> str:  
    pass  
     
     
   class Textbox(ABC):  
    @abstractmethod  
    def operation(self) -> str:  
    pass  
     
     
   class WindowsButton(Button):  
    def operation(self) -> str:  
    return "Windows кнопка"  
     
     
   class HTMLButton(Button):  
    def operation(self) -> str:  
    return "HTML кнопка"  
     
     
   class WindowsTextbox(Textbox):  
    def operation(self) -> str:  
    return "Windows textbox"  
     
     
   class HTMLTextbox(Textbox):  
    def operation(self) -> str:  
    return "HTML textbox"  
     
     
   def client\_code(creator: Dialog) -> None:  
    print(creator.paint(get\_res("Настольный ПК")), end="\n")  
    print(creator.render(), end="")  
    print("\n")  
    print(creator.paint(get\_res("Ноутбук")), end="\n")  
    print(creator.render(), end="")  
    print("\n")  
    print(creator.paint(get\_res("Нетбук")), end="\n")  
    print(creator.render(), end="\n")  
    print("\n")  
     
     
   if \_\_name\_\_ == "\_\_main\_\_":  
    print("КЛИЕНТСКИЙ КОД WINDOWS ДИАЛОГА")  
    client\_code(WindowsDialog())  
    print("\n")  
     
    print("КЛИЕНТСКИЙ КОД WEB ДИАЛОГА")  
    client\_code(WebDialog())
3. **Результат выполнения кода с использованием порождающего паттерна**



****

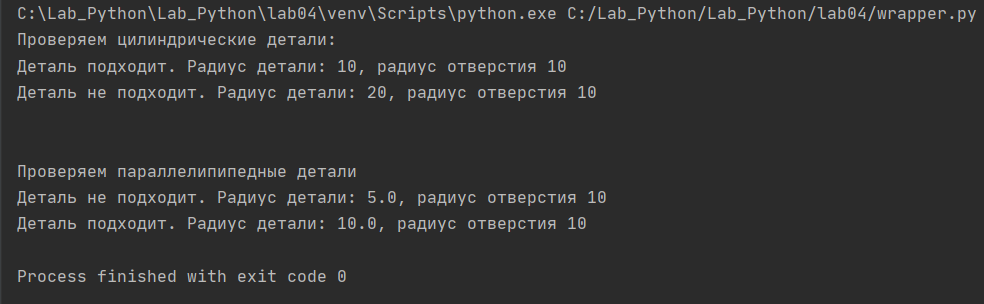
1. **Тесты для порождающего паттерна**
2. from unittest import TestCase, main  
   from unittest.mock import patch  
   from generate import WindowsDialog  
   from generate import WebDialog  
   from generate import WindowsButton  
   from generate import HTMLButton  
     
     
   class AbstractFactoryTestCase(TestCase):  
     
    # проверка верной отрисовки диалога на Windows с разрешением 2560x1600  
    # функцию get\_res делаем Mock-объектом,  
    # т.к. нам важно проверить, чтобы правильно отрисовывалось окно при определенном разрешении,  
    # а не логику функции нахождения разрешения  
    @patch('generate.get\_res', return\_value="2560x1600")  
    def test\_win\_window\_hr(self, get\_res):  
    dialog = WindowsDialog()  
    self.assertEqual("Создание и отрисовка Windows диалога с разрешением 2560x1600",  
    dialog.paint(get\_res("OlegKozinov")))  
     
    # проверка верной отрисовки диалога на Windows с разрешением 1600x1200  
    # функцию get\_res делаем Mock-объектом аналогично прошлому тесту  
    @patch('generate.get\_res', return\_value="1600x1200")  
    def test\_win\_window\_mr(self, get\_res):  
    dialog = WindowsDialog()  
    self.assertEqual("Создание и отрисовка Windows диалога с разрешением 1600x1200",  
    dialog.paint(get\_res("platform")))  
     
    # проверка верной отрисовки диалога на Windows с разрешением 1280x1024  
    # функцию get\_res делаем Mock-объектом аналогично прошлым тестам  
    @patch('generate.get\_res', return\_value="1280x1024")  
    def test\_win\_window\_lr(self, get\_res):  
    dialog = WindowsDialog()  
    self.assertEqual("Создание и отрисовка Windows диалога с разрешением 1280x1024",  
    dialog.paint(get\_res("platform")))  
     
    def test\_win\_render(self):  
    dialog = WindowsDialog()  
    self.assertEqual("Диалог запустился с Windows кнопка и Windows textbox", dialog.render())  
     
    def test\_web\_render(self):  
    dialog = WebDialog()  
    self.assertEqual("Диалог запустился с HTML кнопка и HTML textbox", dialog.render())  
     
    def test\_win\_button(self):  
    dialog = WindowsDialog()  
    button = dialog.create\_button()  
    self.assertEqual("Windows кнопка", button.operation())  
     
    def test\_win\_text(self):  
    dialog = WindowsDialog()  
    button = dialog.create\_button()  
    self.assertEqual("Windows кнопка", button.operation())  
     
    def test\_web\_button(self):  
    dialog = WebDialog()  
    button = dialog.create\_button()  
    self.assertEqual("HTML кнопка", button.operation())  
     
    def test\_web\_text(self):  
    dialog = WebDialog()  
    button = dialog.create\_button()  
    self.assertEqual("HTML кнопка", button.operation())  
     
    # проверка верной отрисовки диалога на Web с разрешением 2560x1600  
    # функцию get\_res делаем Mock-объектом,  
    # т.к. нам важно проверить, чтобы правильно отрисовывалось окно при определенном разрешении,  
    # а не логику функции нахождения разрешения  
    @patch('generate.get\_res', return\_value="2560x1600")  
    def test\_web\_window\_hr(self, get\_res):  
    dialog = WebDialog()  
    self.assertEqual("Создание и отрисовка Web диалога с разрешением 2560x1600",  
    dialog.paint(get\_res("platform")))  
     
    # проверка верной отрисовки диалога на Web с разрешением 1600x1200  
    # функцию get\_res делаем Mock-объектом аналогично прошлому тесту  
    @patch('generate.get\_res', return\_value="1600x1200")  
    def test\_web\_window\_mr(self, get\_res):  
    dialog = WebDialog()  
    self.assertEqual("Создание и отрисовка Web диалога с разрешением 1600x1200",  
    dialog.paint(get\_res("platform")))  
     
    # проверка верной отрисовки диалога на Web с разрешением 1280x1024  
    # функцию get\_res делаем Mock-объектом аналогично прошлым тестам  
    @patch('generate.get\_res', return\_value="1280x1024")  
    def test\_web\_window\_lr(self, get\_res):  
    dialog = WebDialog()  
    self.assertEqual("Создание и отрисовка Web диалога с разрешением 1280x1024",  
    dialog.paint(get\_res("platform")))  
     
     
   if \_\_name\_\_ == '\_\_main\_\_':  
    main()



1. **Структурный паттерн проектирования**

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()

1. **Результат выполнения кода с использованием структурного паттерна**



1. **Тесты для структурного паттерна**

*tests\_wrapper/steps/steps.py*

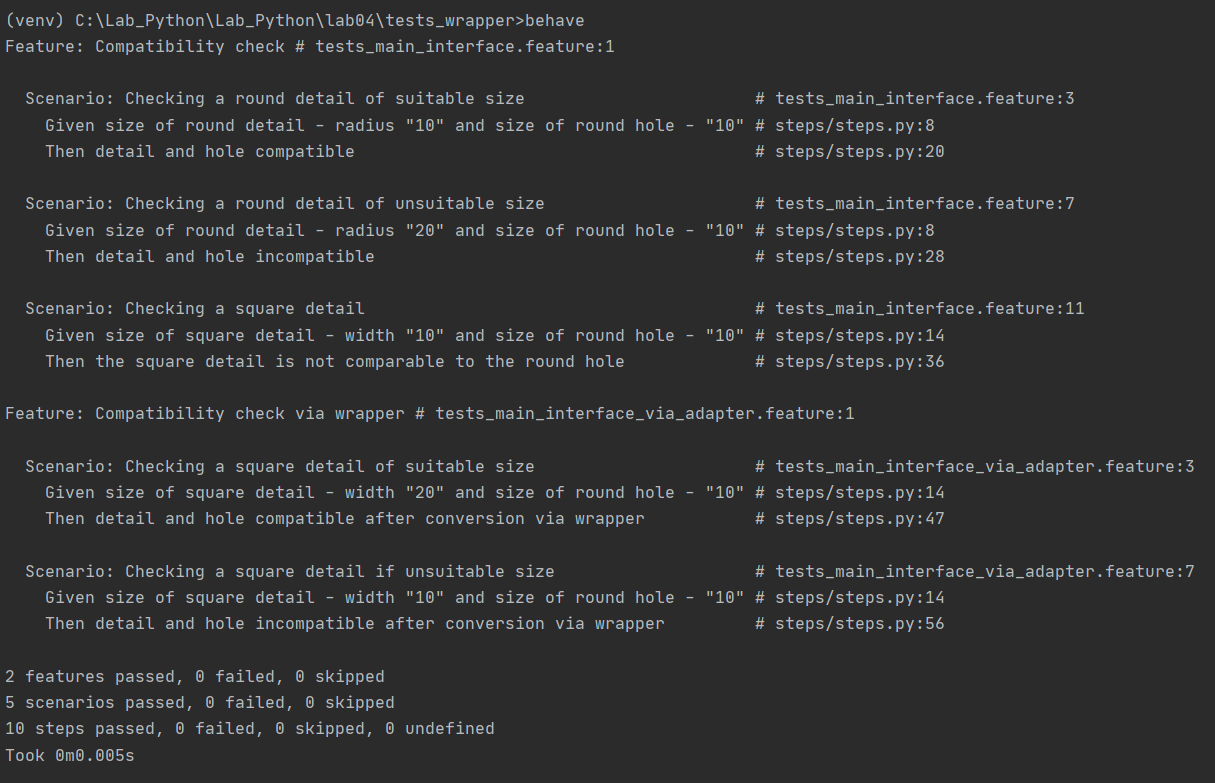
from behave import \*  
from wrapper import RoundDetail  
from wrapper import RoundHole  
from wrapper import SquareDetail  
from wrapper import SquareDetailAdapter  
  
  
@given('size of round detail - radius "{detail\_size}" and size of round hole - "{hole\_radius}"')  
def step(context, detail\_size, 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"

*behav\_test /test\_main.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

*behav\_test /test\_main\_adapter.feature*

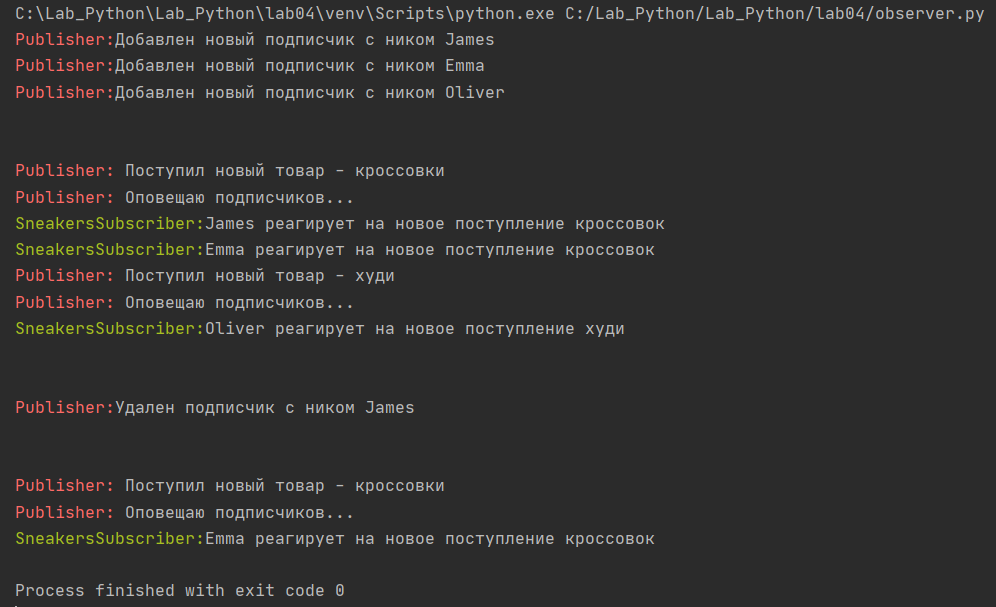
Feature: Compatibility check via wrapper  
  
 Scenario: Checking a square detail of suitable size  
 Given size of square detail - width "20" and size of round hole - "10"  
 Then detail and hole compatible after conversion via wrapper  
  
 Scenario: Checking a square detail if unsuitable size  
 Given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



1. **Поведенческий паттерн проектирования**

from abc import ABC, abstractmethod  
from termcolor import colored  
  
  
# 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 colored("Publisher:", 'red') + f"Добавлен новый подписчик с ником {subscriber.name}"  
  
 def detach(self, subscriber):  
 self.subscribers.remove(subscriber)  
 return colored("Publisher:", 'red') + f"Удален подписчик с ником {subscriber.name}"  
  
 def notify(self):  
 print(colored("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(colored("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 = colored("SneakersSubscriber:", 'green') + f"{self.name} реагирует на новое поступление кроссовок"  
 return react  
 else:  
 return 1  
  
  
# Человек, подписавшиеся на оповещения о поступлении худи  
class HoodiesSubscriber(Subscriber):  
  
 def \_\_init\_\_(self, name):  
 self.name = name  
  
 def update(self, publisher):  
 if publisher.new\_goods == "худи":  
 react = colored("SneakersSubscriber:", 'green') + f"{self.name} реагирует на новое поступление худи"  
 return react  
 else:  
 return 1  
  
  
def client\_code():  
 store = StorePublisher()  
  
 first\_sneakers\_subscriber = SneakersSubscriber("James")  
 print(store.attach(first\_sneakers\_subscriber))  
 second\_sneakers\_subscriber = SneakersSubscriber("Emma")  
 print(store.attach(second\_sneakers\_subscriber))  
 first\_hoodies\_subscriber = HoodiesSubscriber("Oliver")  
 print(store.attach(first\_hoodies\_subscriber))  
  
 print('\n')  
  
 store.goods\_arrival("кроссовки")  
 store.goods\_arrival("худи")  
  
 print('\n')  
  
 print(store.detach(first\_sneakers\_subscriber))  
  
 print('\n')  
  
 store.goods\_arrival("кроссовки")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 client\_code()

1. **Результат выполнения кода с использованием поведенческого паттерна**



1. **Тесты для поведенческого паттерна**

from unittest import TestCase  
from termcolor import colored  
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(colored("SneakersSubscriber:", 'green') +  
 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 = "худи"  
 self.assertEqual(colored("SneakersSubscriber:", 'green') +  
 f"{hoodies\_subscriber.name} реагирует на новое поступление худи",  
 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 = "худи"  
 self.assertEqual(1, sneakers\_subscriber.update(store))

