

Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и
управления»



Отчет
Лабораторная работа № 4
По курсу «Разработка интернет приложений»

ИСПОЛНИТЕЛЬ:

Группа ИУ5-55Б

Петренко С.С.

«18» ноября 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 Моск-объектов

2. Порождающий паттерн проектирования

```
# порождающий паттерн проектирования
# абстрактная фабрика
# предметная область: кроссплатформенные элементы интерфейса,
#                       удовлетворяющие ОС Windows, Mac, Linux

from abc import ABC, abstractmethod

def define_resolution(platform):
    if platform == "Настольный ПК":
        return "2560x1600"
    elif platform == "Ноутбук":
        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("Настольный ПК")))
    print(window.paint(define_resolution("Ноутбук")))
    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())

```

3. Результат выполнения кода с использованием порождающего паттерна

```

C:\Lab_Python\Lab_Python\lab04\venv\Scripts\python.exe C:/Lab_Python/Lab_Python/lab04/abstract_factory.py
Клиентский код на Windows
Создание и отрисовка окна с разрешением 2560x1600 в стиле Windows
Создание и отрисовка окна с разрешением 1600x1200 в стиле Windows
Создание и отрисовка окна с разрешением 1280x1024 в стиле Windows
Отрисовка кнопки в стиле Windows
Отрисовка чек-бокса в стиле Windows
Отрисовка текстового поля в стиле Windows
Отрисовка чек-бокса и Отрисовка кнопки в стиле Windows

Клиентский код на macOS
Создание и отрисовка окна с разрешением 2560x1600 в стиле macOS
Создание и отрисовка окна с разрешением 1600x1200 в стиле macOS
Создание и отрисовка окна с разрешением 1280x1024 в стиле macOS
Отрисовка кнопки в стиле macOS
Отрисовка чек-бокса в стиле macOS
Отрисовка текстового поля в стиле macOS
Отрисовка чек-бокса и Отрисовка кнопки в стиле macOS

Клиентский код на Linux
Создание и отрисовка окна с разрешением 2560x1600 в стиле Linux
Создание и отрисовка окна с разрешением 1600x1200 в стиле Linux
Создание и отрисовка окна с разрешением 1280x1024 в стиле Linux
Отрисовка кнопки в стиле Linux
Отрисовка чек-бокса в стиле Linux
Отрисовка текстового поля в стиле Linux
Отрисовка чек-бокса и Отрисовка кнопки в стиле Linux

Process finished with exit code 0

```

4. Тесты для порождающего паттерна

```

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):

    # 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
прошлomu 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
прошлomu 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())

```

```

✔ Tests passed: 27 of 27 tests - 15 ms
Testing started at 12:26 ...
C:\Lab_Python\Lab_Python\lab04\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1.3\plugins\python-ce\helpers\
Launching unittests with arguments python -m unittest C:/Lab_Python/Lab_Python/lab04/tests_abstract_factory.py in C:\Lab_Python\Lab_Python\lab04

Ran 27 tests in 0.028s

OK

Process finished with exit code 0

```

5. Структурный паттерн проектирования

```

# структурный паттерн проектирования
# адаптер
# предметная область: клиент с помощью исходного интерфейса может
# проверить,
#
#                               подойдет ли цилиндрическая деталь к круглому
отверстию.
#
#                               Появилась возможность обтачивать края
параллелипipedной детали до цилиндрической.
#
#                               Для проверки подойдет ли параллелипipedная деталь
к круглому отверстию
#
#                               необходимо использовать адаптер.

# класс цилиндрических деталей
class RoundDetail:

    def __init__(self, radius):
        self.radius = radius

    def get_radius(self):
        return self.radius

# класс параллелипipedных 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 соответствует
    # 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()

```

6. Результат выполнения кода с использованием структурного паттерна

```

C:\Lab_Python\Lab_Python\lab04\venv\Scripts\python.exe C:/Lab_Python/Lab_Python/lab04/wrapper.py
Проверяем цилиндрические детали:
Деталь подходит. Радиус детали: 10, радиус отверстия 10
Деталь не подходит. Радиус детали: 20, радиус отверстия 10

Проверяем параллелипипедные детали
Деталь не подходит. Радиус детали: 5.0, радиус отверстия 10
Деталь подходит. Радиус детали: 10.0, радиус отверстия 10

Process finished with exit code 0

```

7. 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"

```

tests_wrapper/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_wrapper/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

```

```

(venv) C:\Lab_Python\Lab_Python\lab04\tests_wrapper>behave
Feature: Compatibility check # tests_main_interface.feature:1

    Scenario: Checking a round detail of suitable size                                # tests_main_interface.feature:3
        Given size of round detail - radius "10" and size of round hole - "10" # steps/steps.py:8
        Then detail and hole compatible                                           # steps/steps.py:20

    Scenario: Checking a round detail of unsuitable size                            # tests_main_interface.feature:7
        Given size of round detail - radius "20" and size of round hole - "10" # steps/steps.py:8
        Then detail and hole incompatible                                         # steps/steps.py:28

    Scenario: Checking a square detail                                              # tests_main_interface.feature:11
        Given size of square detail - width "10" and size of round hole - "10" # steps/steps.py:14
        Then the square detail is not comparable to the round hole               # steps/steps.py:36

Feature: Compatibility check via wrapper # tests_main_interface_via_adapter.feature:1

    Scenario: Checking a square detail of suitable size                            # tests_main_interface_via_adapter.feature:3
        Given size of square detail - width "20" and size of round hole - "10" # steps/steps.py:14
        Then detail and hole compatible after conversion via wrapper              # steps/steps.py:47

    Scenario: Checking a square detail if unsuitable size                         # tests_main_interface_via_adapter.feature:7
        Given size of square detail - width "10" and size of round hole - "10" # steps/steps.py:14
        Then detail and hole incompatible after conversion via wrapper            # steps/steps.py:56

2 features passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
10 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.005s

```

8. Поведенческий паттерн проектирования

```

# поведенческий паттерн проектирования
# наблюдатель
# предметная область: магазин одежды делает рассылку подписчикам, при
поступлении нового товара

```

```

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, подписавшийся 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, подписавшийся 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()

```

9. Результат выполнения кода с использованием поведенческого паттерна


```

C:\Lab_Python\Lab_Python\lab04\venv\Scripts\python.exe C:/Lab_Python/Lab_Python/lab04/observer.py
Publisher:Добавлен новый подписчик с ником James
Publisher:Добавлен новый подписчик с ником Emma
Publisher:Добавлен новый подписчик с ником Oliver

Publisher: Поступил новый товар - кроссовки
Publisher: Оповещаю подписчиков...
SneakersSubscriber:James реагирует на новое поступление кроссовок
SneakersSubscriber:Emma реагирует на новое поступление кроссовок
Publisher: Поступил новый товар - худи
Publisher: Оповещаю подписчиков...
SneakersSubscriber:Oliver реагирует на новое поступление худи

Publisher:Удален подписчик с ником James

Publisher: Поступил новый товар - кроссовки
Publisher: Оповещаю подписчиков...
SneakersSubscriber:Emma реагирует на новое поступление кроссовок

Process finished with exit code 0

```

10. 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))

```

```

✔ Tests passed: 6 of 6 tests - 1 ms
Testing started at 12:32 ...
C:\Lab_Python\Lab_Python\lab04\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2020.1.3\plugins\python-ce\h
Launching unittests with arguments python -m unittest C:/Lab_Python/Lab_Python/lab04/tests_observer.py in C:\Lab_Python\Lab_Python\lab04

Ran 6 tests in 0.003s

OK

Process finished with exit code 0

```