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**
   1.1 Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.
   1.2 Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
      - TDD – фреймворк
      - BDD – фреймворк
      - Создание Mock-объектов
2. **Порождающий паттерн проектирования**

```
3. 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())
```

4. Результат выполнения кода с использованием порождающего паттерна

```
КЛИЕНТСКИЙ КОД WINDOWS ДИАЛОГА
Создание и отрисовка Windows диалога с разрешением 2560x1600
Создание и отрисовка Windows кнопка
Создание и отрисовка Windows textbox
Диалог запустился с Windows кнопка и Windows textbox

Создание и отрисовка Windows диалога с разрешением 1600x1200
Создание и отрисовка Windows кнопка
Создание и отрисовка Windows textbox
Диалог запустился с Windows кнопка и Windows textbox

Создание и отрисовка Windows диалога с разрешением 1280x1024
Создание и отрисовка Windows кнопка
Создание и отрисовка Windows textbox
Диалог запустился с Windows кнопка и Windows textbox
```

```
КЛИЕНТСКИЙ КОД WEB ДИАЛОГА
Создание и отрисовка Web диалога с разрешением 2560x1600
Создание и отрисовка HTML кнопка
Создание и отрисовка HTML textbox
Диалог запустился с HTML кнопка и HTML textbox

Создание и отрисовка Web диалога с разрешением 1600x1200
Создание и отрисовка HTML кнопка
Создание и отрисовка HTML textbox
Диалог запустился с HTML кнопка и HTML textbox

Создание и отрисовка Web диалога с разрешением 1280x1024
Создание и отрисовка HTML кнопка
Создание и отрисовка HTML textbox
Диалог запустился с HTML кнопка и HTML textbox
```

5. Тесты для порождающего паттерна

```
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()
```

```
............
----------------------------------------------------------------------
Ran 12 tests in 0.006s

OK
```

7. 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()
```

8. **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
```

9. 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
```

```
(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
```

10.Поведенческий паттерн проектирования

```python
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()
```

11. 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
```

12. 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))
```