Московский государственный технический университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5

Отчёт по

лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python.»

Подготовил:

Алпеев Владислав Сергеевич

Группа ИУ5-54Б

Москва
2020

1. Описание задания.

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   - TDD - фреймворк.
   - BDD - фреймворк.
   - Создание Mock-объектов.

2. Текст программы.

webElement.py

```
from abc import ABC, abstractmethod, abstractproperty


#Абстрактный класс посредника
class Mediator(ABC):
    def notify(self, sender: object, event: str) -> None:
        pass


#Абстрактный класс строителя
class WebElementBuilder(ABC):

    @abstractproperty
    def product(self):
        pass

    @abstractmethod
    def setName(self):
        pass

    @abstractmethod
    def introduce_height(self):
        pass

    @abstractmethod
    def introduce_width(self):
        pass
```

```
    @abstractmethod
    def introduce_visibility(self):
        pass

    @abstractmethod
    def introduce_connectivity(self):
        pass

    @abstractmethod
    def introduce_action(self):
        pass

    @abstractmethod
    def introduce_link(self):
        pass

    @abstractmethod
    def introduce_text(self):
        pass


#Конкретный класс строителя
class ConcreteWebElementBuilder(WebElementBuilder):
    def __init__(self):
        self.reset()

    def reset(self):
        self._product = WebElement()
        self._mediator = None

    @property
    def product(self):
        product = self._product
        self.reset()
        return product

    def setName(self, name):
        self._product.nameOfElement = name
```

```
    def introduce_height(self, value):
        self._product.add("height", value)

    def introduce_width(self, value):
        self._product.add("width", value)

    def introduce_visibility(self, value):
        self._product.add("visibility", value)

    def introduce_connectivity(self, value):
        self._product.add("connectivity", True)
        self._product.add("mainElement", value)

    def introduce_action(self, name, value):
        self._product.add("action", (name, value))

    def introduce_link(self, value):
        self._product.add("link", value)

    def introduce_text(self, value):
        self._product.add("text", value)


#Класс веб элемента
class WebElement():
    def __init__(self):
        self.parts = {}
        self._name = None

    @property
    def nameOfElement(self):
        return self._name

    @nameOfElement.setter
    def nameOfElement(self, nameOfElement):
        self._name = nameOfElement

    @property
    def mediator(self) -> Mediator:
        return self._mediator
```

```
    @mediator.setter
    def mediator(self, mediator: Mediator):
        self._mediator = mediator

    @nameOfElement.getter
    def nameOfElement(self):
        return self._name

    def add(self, partName, partValue):
        self.parts.update({partName: partValue})

    def printParts(self):
        print("\033[33m\n {} \n\033[37m".format(self._name))
        for i, j in self.parts.items():
            print("{} = {}".format(i, j))

    def makeAction(self):
        self._mediator.notify(self, self.parts["action"][0])
        func = self.parts["action"][1]
        func()


#Конкретная реализация посредника
class ConcreateMediator(Mediator):
    def __init__(self, webElement1: WebElement, webElement2: WebElement):
        self._webElement1 = webElement1
        self._webElement1.mediator = self
        self._webElement2 = webElement2
        self._webElement2.mediator = self

    def notify(self, sender: object, event: str):
        if (event == "clicked") and (sender.nameOfElement == "<button>"):
            print("\033[35m\nНажата кнопка, запрошено обновление label\033[37m\n")
            self._webElement2.makeAction()
        elif (event == "updated") and (sender.nameOfElement == "<label>"):
            print("\033[35m\nПосле нажатия кнопки label был обновлен\033[37m\n")
```

director.py

```
from webElement import WebElementBuilder, ConcreteWebElementBuilder
```

```
#Класс директора
class WebDirector():
    def __init__(self, builder: WebElementBuilder):
        self._builder = builder

    def make_label(self):
        self._builder.setName("<label>")
        self._builder.introduce_text("I am <label>")
        self._builder.introduce_visibility(True)
        self._builder.introduce_action("updated", labelUpdate)

    def make_button(self):
        self._builder.setName("<button>")
        self._builder.introduce_height(30)
        self._builder.introduce_width(120)
        self._builder.introduce_visibility(True)
        self._builder.introduce_text("I am <button>")
        self._builder.introduce_action("clicked", buttonClick)

    def make_link(self):
        self._builder.setName("<a>")
        self._builder.introduce_text("I am <a>")
        self._builder.introduce_visibility(True)
        self._builder.introduce_link("https://smth.com")

    def make_radioButton(self):
        self._builder.setName("<radio>")
        self._builder.introduce_text("I am <radio>")
        self._builder.introduce_connectivity("previousElem")
        self._builder.introduce_visibility(True)

def labelUpdate():
    print("\n\033[33mДополнительная логика для лейбла\033[33m\n")

def buttonClick():
    print("\n\033[33mДополнительная логика кнопки\033[33m\n")
```

TDD_test.py

```
from programmingPatterns.webElement import ConcreteWebElementBuilder
```

```
from programmingPatterns.director import WebDirector, labelUpdate,
buttonClick

if __name__ == "__main__":

    builder = ConcreteWebElementBuilder()
    director = WebDirector(builder)

    director.make_label()
    label = builder.product.printParts()
    assert label == {'text': 'I am <label>', 'visibility': True, 'action':
('updated', labelUpdate)}, "Неправильно создан элемент label"

    director.make_button()
    button = builder.product.printParts()
    assert button == {'height': 30, 'width': 120, 'visibility': True,
'text': 'I am <button>', 'action': ('clicked', buttonClick)}, "Неправильно
создан элемент button"

    director.make_link()
    link = builder.product.printParts()
    assert link == {'text': 'I am <a>', 'visibility': True, 'link':
'https://smth.com'}, "Неправильно создан элемент label"

    director.make_radioButton()
    radioButton = builder.product.printParts()
    assert radioButton == {'text': 'I am <radio>', 'connectivity': True,
'mainElement': 'previousElem', 'visibility': True}, "Неправильно создан
элемент label"
```

BDD_test.py

```
from abc import ABC
from programmingPatterns.webElement import ConcreateMediator,
ConcreteWebElementBuilder
from programmingPatterns.director import WebDirector

if __name__ == "__main__":
    builder = ConcreteWebElementBuilder()
    director = WebDirector(builder)
```

```
director.make_button()
button = builder.product

director.make_label()
label = builder.product

mediator = ConcreateMediator(button, label)

button.makeAction()
```

3. Экранные формы с примерами выполнения программы.