Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет-приложений»

Отчет по лабораторной работе №4

Выполнила:

студент группы ИУ5-53Б

Латыпова К.Н.

Москва, 2020 г.

1. 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:

a. TDD - фреймворк.
b. BDD - фреймворк.
c. Создание Mock-объектов.

2. Текст программы

main.py:

```
from abc import ABC, abstractmethod, abstractproperty


class Box(ABC):

    @abstractproperty
    def box(self):

        pass

    @abstractmethod
    def add_pomade(self):
        pass

    @abstractmethod
    def add_ink(self):
        pass

    @abstractmethod
    def add_eyeshadow(self):
        pass

    @abstractmethod
    def add_blush(self):
        pass

class Box1(Box):
    """Конкретный строитель, строящий бокс первого типа."""
    def __init__(self):
        self.reset()

    def reset(self):
        self._box = Boxx()

    @property
    def box(self):
        box = self._box
```

```
        return box

    def add_pomade(self):
        self._box.add("Джинсы", 2200)

    def add_ink(self):
        self._box.add("Футболка", 900)

    def add_eyeshadow(self):
        self._box.add("Кроссовки \"Зима\"", 6450)

    def add_blush(self):
        self._box.add("Куртка", 7800)

    def add_all(self):
        self.add_pomade()
        self.add_ink()
        self.add_eyeshadow()
        self.add_blush()


class Box2(Box):
    """Конкретный строитель, строящий бокс второго типа."""
    def __init__(self):
        self.reset()

    def reset(self):
        self._box = Boxx()

    @property
    def box(self):
        box = self._box
        return box

    def add_pomade(self):
        self._box.add("Джинсы", 1200)

    def add_ink(self):
        self._box.add("Футболка", 1050)

    def add_eyeshadow(self):
        self._box.add("Кроссовки \"Лето\"", 4999)

    def add_blush(self):
        self._box.add("Куртка", 6560)

    def add_all(self):
        self.add_pomade()
        self.add_ink()
        self.add_eyeshadow()
        self.add_blush()


class Boxx():
    def __init__(self):
        self.box = []
        self.sum = 0

    def add(self, dish, price):
        self.box.append(dish)
        self.sum += price

    def list_box(self):
        return f"{', '.join(self.box)}"
```

```
    def get_sum(self):
        return self.sum


if __name__ == '__main__':
    print('Заказ №1 ')
    order = Box1()
    order.add_pomade()
    order.add_ink()
    order.add_blush()
    print(order.box.list_box())

    print('\nЗаказ №2 ')
    order.reset()
    order.add_ink()
    order.add_eyeshadow()
    print(order.box.list_box())

    print('\nЗаказ №3 ')
    order = Box2()
    order.add_all()
    print(order.box.list_box())
```

main1.py:

```
from abc import ABC, abstractmethod
from main import Box1, Box2


class Component(ABC):
    """
        Базовый класс Компонент объявляет общие операции как для простых, так
и для
        сложных объектов структуры.
    """
    @property
    def parent(self):
        return self._parent

    @parent.setter
    def parent(self, parent):
        self._parent = parent

    def add(self, component):
        pass

    def remove(self, component):
        pass

    def is_composite(self):
        return False

    @abstractmethod
    def operation(self):
        pass

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

```python
class Leaf(Component):
    """Конечный объект, не имеющий вложенных."""
    def __init__(self, value, price):
        self._value = value
        self._price = price

    def operation(self):
        return self._value

    def get_price(self):
        return self._price


class Composite(Component):
    """Объект, имеющий вложенные объекты."""
    def __init__(self, name):
        self._children = []
        self._name = name

    def add(self, component):
        self._children.append(component)
        component.parent = self

    def remove(self, component):
        self._children.remove(component)
        component.parent = None

    def is_composite(self):
        return True

    def operation(self):
        results = []
        for child in self._children:
            results.append(child.operation())
        return self._name+f"({'+'.join(results)})"

    def get_price(self):
        count = 0
        for child in self._children:
            count += child.get_price()
        return count

    def accept(self, visitor1):
        pass


def client_code(component):
    print(f"Box: {component.operation()}")
    print(f'Общая стоимость: {component.get_price()}', end='\n\n')


if __name__=='__main__':
    catalog = Composite('Каталог')
    blush = Composite('Кроссовки')
    blush.add(Leaf('Зима', 6450))
    blush.add(Leaf('Лето', 4999))
    boxes = Composite('Боксы')
    box1 = Box1()
    box1.add_all()
    box2 = Box2()
    box2.add_all()
    boxes.add(Leaf(box1.box.list_box(), box1.box.get_sum()))
    boxes.add(Leaf(box2.box.list_box(), box2.box.get_sum()))
```

```
    paper = Leaf('Носки', 150)

    catalog.add(blush)
    catalog.add(boxes)
    catalog.add(paper)

    client_code(catalog)
    client_code(boxes)
    client_code(paper)
    client_code(blush)
```

main2.py:

```
from main1 import Composite, Component, Leaf, client_code
from main import Box1, Box2
from abc import ABC, abstractmethod


class ComponentNew(Component):
    """
        Интерфейс Компонента объявляет метод accept, который в качестве
аргумента
        может получать любой объект, реализующий интерфейс посетителя.
    """
    @abstractmethod
    def accept(self, visitor):
        pass


class CompositeNew(Composite, ComponentNew):
    def accept(self, visitor):
        visitor.visit_component(self)


class Visitor(ABC):
    @abstractmethod
    def visit_component(self, element):
        pass


class Visitor1(Visitor):
    def visit_component(self, element):
        print('Стоимость: {}'.format(element.get_price()))


class Visitor2(Visitor):
    def visit_component(self, element):
        client_code(element)


if __name__ == '__main__':
    catalog = CompositeNew('Каталог')
    blush = Composite('Кроссовки')
    blush.add(Leaf('Зима', 6450))
    blush.add(Leaf('Лето', 4999))
    boxes = Composite('Боксы')
    box1 = Box1()
    box1.add_all()
    box2 = Box2()
    box2.add_all()
    boxes.add(Leaf(box1.box.list_box(), box1.box.get_sum()))
```

```
    boxes.add(Leaf(box2.box.list_box(), box2.box.get_sum()))
    paper = Leaf('Носки', 150)

    catalog.add(blush)
    catalog.add(boxes)
    catalog.add(paper)

    visitor1 = Visitor1()
    visitor2 = Visitor2()
    print("Первый посетитель:")
    catalog.accept(visitor1)
    print("\nВторой посетитель:")
    catalog.accept(visitor2)
```

TDD.py:

```
from main2 import *
import unittest


class SummaTest(unittest.TestCase):
    def test_summa_menu(self):
        catalog = Composite('Каталог')
        blush = Composite('Кроссовки')
        blush.add(Leaf('Зима', 6450))
        blush.add(Leaf('Лето', 4999))
        boxes = Composite('Боксы')
        box1 = Box1()
        box1.add_all()
        box2 = Box2()
        box2.add_all()
        boxes.add(Leaf(box1.box.list_box(), box1.box.get_sum()))
        boxes.add(Leaf(box2.box.list_box(), box2.box.get_sum()))
        paper = Leaf('Носки', 150)

        catalog.add(blush)
        catalog.add(boxes)
        catalog.add(paper)

        visitor1 = Visitor1()
        self.assertEqual(catalog.accept(visitor1),
'visitor_for_composite_new', "Should be 'visitor_for_composite_new'")


if __name__ == "__main__":
    unittest.main()
```

BDD.py:

```
from radish import given, when, then


@given("I have the component {component1: g}")
def have_component(step, component1):
    step.context.component1 = component1


@when("I get price from them")
def get_price_component(step):
```

```
    step.context.result = step.context.component1.get_price()


@then("I expect the result to be {result: g}")
def expect_result(step, result):
    assert step.context.result == result
```

unitest.py:

```
from main import Box1, Box2
import unittest


class SummaTest(unittest.TestCase):
    def test_sum_Box1(self):
        order = Box1()
        order.add_all()
        self.assertEqual(order.box.get_sum(), 17350, "Should be 17350")

    def test_sum_Box2(self):
        order = Box2()
        order.add_all()
        self.assertEqual(order.box.get_sum(), 13809, "Should be 13809")


if __name__ == "__main__":
    unittest.main()
```

3. Экранные формы с примерами выполнения программы

main.py:

```
Заказ №1
Джинсы, Футболка, Куртка

Заказ №2
Футболка, Кроссовки "Зима"

Заказ №3
Джинсы, Футболка, Кроссовки "Лето", Куртка

Process finished with exit code 0
```

main1.py:

```
Box: Каталог(Кроссовки(Зима+Лето)+Боксы(Джинсы, Футболка, Кроссовки "Зима", Куртка+Джинсы, Футболка, Кроссовки "Лето", Куртка)+Носки)
Общая стоимость: 42758

Box: Боксы(Джинсы, Футболка, Кроссовки "Зима", Куртка+Джинсы, Футболка, Кроссовки "Лето", Куртка)
Общая стоимость: 31159

Box: Носки
Общая стоимость: 150

Box: Кроссовки(Зима+Лето)
Общая стоимость: 11449


Process finished with exit code 0
```

main2.py:

```
Первый посетитель:
Стоимость: 42758

Второй посетитель:
Box: Каталог(Кроссовки(Зима+Лето)+Боксы(Джинсы, Футболка, Кроссовки "Зима", Куртка+Джинсы, Футболка, Кроссовки "Лето", Куртка)+Носки)
Общая стоимость: 42758


Process finished with exit code 0
```

unitest.py:

```
Ran 2 tests in 0.004s

OK

Process finished with exit code 0
```