**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет “Информатика и системы управления”

Кафедра ИУ-5 “Системы обработки информации и управления”

Курс “Разработка интернет-приложений”

Отчет по лабораторной работе №4

## “ Шаблоны проектирования и модульное тестирование в Python”

Выполнила:

Студентка группы ИУ5-53Б

Желтова Александра

Москва, 2020

**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.](https://refactoring.guru/ru/design-patterns/catalog)

2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

**Цель лабораторной работы:**изучение основ языка Python.

**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\_eyeliner(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("Помада", 1200)

def add\_ink(self):

self.\_box.add("Тушь", 500)

def add\_eyeliner(self):

self.\_box.add("Подводка", 3450)

def add\_blush(self):

self.\_box.add("Пудра", 600)

def add\_all(self):

self.add\_pomade()

self.add\_ink()

self.add\_eyeliner()

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("Помада", 900)

def add\_ink(self):

self.\_box.add("Тушь", 1050)

def add\_eyeliner(self):

self.\_box.add("Подводка", 799)

def add\_blush(self):

self.\_box.add("Пудра", 810)

def add\_all(self):

self.add\_pomade()

self.add\_ink()

self.add\_eyeliner()

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('Предложение №1 ')

order = Box1()

order.add\_pomade()

order.add\_blush()

print(order.box.list\_box())

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print('\nПредложение №2 ')

order.reset()

order.add\_ink()

order.add\_eyeliner()

print(order.box.list\_box())

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

print('\nПредложение №3 ')

order = Box2()

order.add\_all()

print(order.box.list\_box())

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

**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

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('Подводка')

blush.add(Leaf('Черная', 1750))

boxes = Composite('Наборы')

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('Средство для снятия макияжа', 250)

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('Подводка')

blush.add(Leaf('Черная', 1750))

boxes = Composite('Наборы')

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('Средство для снятия макияжа', 250)

catalog.add(blush)

catalog.add(boxes)

catalog.add(paper)

visitor1 = Visitor1()

visitor2 = Visitor2()

print("Первый клиент:")

catalog.accept(visitor1)

print("\nВторой клиент:")

catalog.accept(visitor2)

**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(), 5750, "Should be 5750")

def test\_sum\_Box2(self):

order = Box2()

order.add\_all()

self.assertEqual(order.box.get\_sum(), 3559, "Should be 3559")

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

**TDD.py:**

from main2 import \*

import unittest

class SummaTest(unittest.TestCase):

def test\_summa\_menu(self):

catalog = Composite('Каталог')

blush = Composite('Подводка')

blush.add(Leaf('Черная', 1750))

boxes = Composite('Наборы')

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('Средство для снятия макияжа', 250)

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()

**Экранные формы:**

 

