Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ	**«Информатики и систем управления»**
КАФЕДРА	Системы обработки информации и управления

Дисциплина «Разработка Интернет-Приложений»

ЛАБОРАТОРНАЯ РАБОТА №4

Шаблоны проектирования и модульное тестирование в Python

Студент	Сахарова Е. К. ИУ5-52Б
Преподаватель	Гапанюк Ю. Е.

Цель лабораторной работы: изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

Задание:

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.

Код программы:

Предметная область для выполнения задания – Торт.

builder.py

```
from abc import ABCMeta, abstractstaticmethod


class ICakeBuilder(metaclass=ABCMeta):
    """Интерфейс строителя"""

    @abstractstaticmethod
    def set_foundation(self, value):
        """Задает основу для торта"""

    @abstractstaticmethod
    def set_shape_type(self, value):
        """Задает форму для торта"""

    @abstractstaticmethod
    def set_filling(self, value):
        """Задает тип начинки"""

    @abstractstaticmethod
    def set_number_tiers(self, value):
        """Задает количество ярусов торта"""

    @abstractstaticmethod
    def get_result(self):
        """Возвращает торт"""


class CakeBuilder(ICakeBuilder):
    """Конкретный строитель"""

    def __init__(self):
        self.cake = Cake()

    def set_foundation(self, value):
        self.cake.foundation = value
        return self

    def set_shape_type(self, value):
        self.cake.shape_type = value
        return self

    def set_filling(self, value):
        self.cake.filling = value
        return self

    def set_number_tiers(self, value):
        self.cake.tiers = value
        return self

    def get_result(self):
        return self.cake


class Cake:
    """Торт"""
```

```
    def __init__(self, foundation='бисквитный', shape_type='круга',
filling='шоколадная', tiers=1):
        # бисквитный, песочный, слоеный, шоколадный
        self.foundation = foundation
        # круг, овал, квадрат, сердце, ромб, звезда
        self.shape_type = shape_type
        # шоколадная, клубничная, вишневая, йогуртовая, ореховая
        self.filling = filling
        self.tiers = tiers

    def __str__(self):
        return 'Это {0} торт в форме {1} с {2} начинкой и {3} ярусами.'.format(
            self.foundation, self.shape_type, self.filling, self.tiers
        )


class YogurtDirector:
    @staticmethod
    def construct():
        return CakeBuilder()\
            .set_foundation('бисквитный')\
            .set_shape_type('овала')\
            .set_filling('йогуртовой')\
            .set_number_tiers(3)\
            .get_result()


class StrawberryDirector:
    @staticmethod
    def construct():
        return CakeBuilder()\
            .set_foundation('бисквитный')\
            .set_shape_type('сердца')\
            .set_filling('клубничной')\
            .set_number_tiers(1)\
            .get_result()


class ChocolateDirector:
    @staticmethod
    def construct():
        return CakeBuilder()\
            .set_foundation('шоколадный')\
            .set_shape_type('квадрата')\
            .set_filling('ореховой')\
            .set_number_tiers(1)\
            .get_result()


if __name__ == '__main__':
    YOGURT_CAKE = YogurtDirector.construct()
    print(YOGURT_CAKE)
    STRAWBERRY_CAKE = StrawberryDirector.construct()
    print(STRAWBERRY_CAKE)
    CHOCOLATE_CAKE = ChocolateDirector.construct()
    print(CHOCOLATE_CAKE)
```

test-tdd.py

```
import unittest
import sys
import os
# from github.lab4.builder import *
from builder import *


sys.path.append(os.getcwd())


class TestFoundation(unittest.TestCase):
    def test_yogurt_cake_returns_string(self):
        self.assertEqual('{}'.format(YogurtDirector.construct()),
                         'Это бисквитный торт в форме овала с йогуртовой начинкой и 3
ярусами.')

    def test_strawberry_cake_returns_string(self):
        self.assertEqual('{}'.format(StrawberryDirector.construct()),
                         'Это бисквитный торт в форме сердца с клубничной начинкой и
1 ярусами.')

    def test_chocolate_cake_returns_string(self):
        self.assertEqual('{}'.format(ChocolateDirector.construct()),
                         'Это шоколадный торт в форме квадрата с ореховой начинкой и
1 ярусами.')


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

bridge.py

```
from abc import ABCMeta, abstractmethod


class Decoration(metaclass=ABCMeta):
    """Декорирование"""

    @abstractmethod
    def decorating_strawberry_cake(self, items):
        """Клубничный торт"""

    @abstractmethod
    def decorating_chocolate_cake(self, items):
        """Шоколадный торт"""


class Candle(Decoration):
    """Декорирование свечами"""

    def decorating_strawberry_cake(self, items):
        return f"Клубничный торт с {items} свечами."

    def decorating_chocolate_cake(self, items):
        return f"Шоколадный торт с {items} свечами."


class Flower(Decoration):
```

```python
    """Декорирование цветами"""

    def decorating_strawberry_cake(self, items):
        return f"Клубничный торт с {items} цветами."

    def decorating_chocolate_cake(self, items):
        return f"Шоколадный торт с {items} цветами."


class Cake(metaclass=ABCMeta):
    """Торт"""

    @abstractmethod
    def __init__(self, decoration):
        self.decoration = decoration

    @abstractmethod
    def display_description(self):
        """Выводит информацию"""

    @abstractmethod
    def add_objects(self):
        """Добавляет объект декора"""


class Strawberry(Cake):
    def __init__(self, decoration, objects=0):
        super().__init__(decoration)
        self.objects = objects

    def display_description(self):
        return self.decoration.decorating_strawberry_cake(self.objects)

    def add_objects(self, new_objects):
        self.objects += new_objects


class Chocolate(Cake):
    def __init__(self, decoration, objects=0):
        super().__init__(decoration)
        self.objects = objects

    def display_description(self):
        return self.decoration.decorating_chocolate_cake(self.objects)

    def add_objects(self, new_objects):
        self.objects += new_objects


if __name__ == '__main__':
    candle = Candle()
    flower = Flower()

    chocolate1 = Chocolate(candle, 20)
    chocolate2 = Chocolate(flower, 2)
    strawberry1 = Strawberry(candle, 4)
    strawberry2 = Strawberry(flower, 50)

    print(strawberry1.display_description())
    strawberry1.add_objects(1)
    print(strawberry1.display_description())
```

```
    print(chocolate1.display_description())
    print(chocolate2.display_description())
    print(strawberry2.display_description())
```

cake.feature

```
Feature: Test Bridge Design Pattern Functionality
    Scenario: Strawberry Cake with 8 Candles
        Given Bridge app is run
        When I input "4" candles to put on the cake
        Then I get result "Клубничный торт с 4 свечами."
```

steps.py

```
from behave import given, when, then
from bridge import *


@given(u'Bridge app is run')
def step_impl(context):
    print(u'STEP: Given Bridge app is run')
    pass


@when(u'I input "{inp}" candles to put on the cake')
def step_impl(context, inp):
    print(u'STEP: When I input "{}" to put on the cake'.format(inp))
    context.result = '{}'.format(Strawberry(Candle(), inp).display_description())


@then(u'I get result "{out}"')
def step_impl(context, out):
    print(u'STEP: Then I get result "{}"'.format(out))
    assert context.result == out, 'Expected {}, got {}'.format(out, context.result)
```

chain_of_responsibility.py

```
from abc import ABCMeta, abstractstaticmethod


class ICake(metaclass=ABCMeta):
    """Интерфейс обработки запросов"""

    @abstractstaticmethod
    def set_successor(successor):
        """Установка следующего обработчика в цепочке"""

    @abstractstaticmethod
    def handle(amount):
        """Обработка события"""


class BigCake(ICake):
    """Конкретный торт
    Выдает большой торт весом 5 кг, если применимо,
    в противном случае продолжает """

    def __init__(self):
```

```
        self._successor = None

    def set_successor(self, successor):
        """Установка преемника"""
        self._successor = successor

    def handle(self, amount):
        """Обработка торта"""
        if amount >= 5:
            num = amount // 5
            remainder = amount % 5
            print(f'Больших тортов весом 5 кг -- {num}')
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)


class MediumCake(ICake):
    """Конкретный торт
    Выдает большой торт весом 3 кг, если применимо,
    в противном случае продолжает """

    def __init__(self):
        self._successor = None

    def set_successor(self, successor):
        """Установка преемника"""
        self._successor = successor

    def handle(self, amount):
        """Обработка торта"""
        if amount >= 3:
            num = amount // 3
            remainder = amount % 3
            print(f'Средних тортов весом 3 кг -- {num}')
            if remainder != 0:
                self._successor.handle(remainder)
        else:
            self._successor.handle(amount)


class SmallCake(ICake):
    """Конкретный торт
    Выдает большой торт весом 1 кг, если применимо,
    в противном случае продолжает """

    def __init__(self):
        self._successor = None

    def set_successor(self, successor):
        """Установка преемника"""
        self._successor = successor

    def handle(self, amount):
        """Обработка торта"""
        if amount >= 1:
            num = amount // 1
            remainder = amount % 1
            print(f'Маленьких тортов весом 1 кг -- {num}')
            if remainder != 0:
                self._successor.handle(remainder)
```

```
        else:
            self._successor.handle(amount)


class CakeChain:
    """Связь тортов"""

    def __init__(self):
        self.chain1 = BigCake()
        self.chain2 = MediumCake()
        self.chain3 = SmallCake()

        self.chain1.set_successor(self.chain2)
        self.chain2.set_successor(self.chain3)


if __name__ == '__main__':
    CAKES = CakeChain()
    AMOUNT = int(input('Введите желаемую сумму всех тортов: '))
    if AMOUNT < 1:
        print('Введите положительное число')
        exit()
    CAKES.chain1.handle(AMOUNT)
    print('Готово!')
```

test_mock.py

```
import unittest
from unittest.mock import patch
import io
# from github.lab4.chain_of_responsibility import *
from chain_of_responsibility import *


class TestFoundation(unittest.TestCase):

    @patch('sys.stdout', new_callable=io.StringIO)
    def test_foo_one(self, mock_stdout):
        CakeChain().chain1.handle(6)
        assert mock_stdout.getvalue() == 'Больших тортов весом 5 кг -- 1\nМаленьких
тортов весом 1 кг -- 1\n'

    def test_foo_two(self):
        with patch('sys.stdout', new=io.StringIO()) as fake_stdout:
            CakeChain().chain1.handle(5)

        assert fake_stdout.getvalue() == 'Больших тортов весом 5 кг -- 1\n'


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

Экранные формы с примерами выполнения программы:

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab4/builder.py
Это бисквитный торт в форме овала с йогуртовой начинкой и 3 ярусами.
Это бисквитный торт в форме сердца с клубничной начинкой и 1 ярусами.
Это шоколадный торт в форме квадрата с ореховой начинкой и 1 ярусами.

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab4/tests/test_tdd.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab4/bridge.py
Клубничный торт с 4 свечами.
Клубничный торт с 5 свечами.
Шоколадный торт с 20 свечами.
Шоколадный торт с 2 цветами.
Клубничный торт с 50 цветами.

Process finished with exit code 0
```

```
C:\Users\sakha\Downloads\Универ\RIP\github\lab4\features>behave cake.feature
Feature: Test Bridge Design Pattern Functionality # cake.feature:1

  Scenario: Strawberry Cake with 8 Candles           # cake.feature:2
    Given Bridge app is run                          # ../steps/steps.py:5
    When I input "4" candles to put on the cake      # ../steps/steps.py:11
    Then I get result "Клубничный торт с 4 свечами." # ../steps/steps.py:17

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
3 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.001s
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab4/chain_of_responsibility.py
Введите желаемую сумму всех тортов: 73
Больших тортов весом 5 кг -- 14
Средних тортов весом 3 кг -- 1
Готово!

Process finished with exit code 0
```

```
C:\Programs\Python\python.exe C:/Users/sakha/Downloads/Универ/RIP/github/lab4/tests/test_mock.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK

Process finished with exit code 0
```