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 «Радиотехнический»

КАФЕДРА ИУ-5 «Системы обработки информации и управления»

Отчет по лабораторной работе № 4 по курсу

Разработка интернет-приложений

Тема работы: "Шаблоны проектирования и модульное тестирование в Python."

Выполнил: Лисин. А. В.
Группа: РТ5-51Б

Дата выполнения: «11» декабря 2020 г.

Подпись:_____

Проверил: Гапанюк Ю. Е.

Дата проверки: «11» декабря 2020 г.

Подпись:_____

Москва, 2020 г.

Содержание

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

Описание задания:

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:

- TDD - фреймворк.
- BDD - фреймворк.
- Создание Mock-объектов.

Ход работы:

Текст context_abstract.py:

```
from abc import ABC, abstractmethod


# Абстрактный класс для контекстов ("Фабрика")
class SetContext(ABC):

    #Внесение стратегии отбора элементов множества как поля класса контекста ("Мост")
    def SetStrategy(self, newStrategy):
        self.strategy = newStrategy

    def GetStrategy(self):
        return self.strategy

    @abstractmethod
    def ExecuteOperation(self, set1, set2):
        pass
```

Текст context_difference.py:

```
from factory.context_abstract import SetContext
import random


# Контекст работы с операций разности множеств
class SetDifferenceContext(SetContext):

    def __init__(self, initialStrategy = None):
        self.strategy = initialStrategy

    def SetStrategy(self, newStrategy):
        SetContext.SetStrategy(self, newStrategy)

    def GetStrategy(self):
        SetContext.GetStrategy(self)
```

```
    def ExecuteOperation(self, set1, set2):

        # Для демонстрации необходимости mock-объектов
        def genRandom(amountOfNums, minNum, maxNum):
            for i in range(amountOfNums):
                yield random.randint(minNum, maxNum)

        set1 = set([i for i in genRandom(1000000, -5, 5)])
        filteredSet1 = self.strategy.filteredSet(set1)
        filteredSet2 = self.strategy.filteredSet(set2)
        return filteredSet1 - filteredSet2
```

Текст context_intersection.py:

```
from factory.context_abstract import SetContext


# Контекст работы с операций пересечения множеств
class SetIntersectionContext(SetContext):

    def __init__(self, initialStrategy = None):
        self.strategy = initialStrategy

    def SetStrategy(self, newStrategy):
        SetContext.SetStrategy(self, newStrategy)

    def GetStrategy(self):
        SetContext.GetStrategy(self)

    def ExecuteOperation(self, set1, set2):
        filteredSet1 = self.strategy.filteredSet(set1)
        filteredSet2 = self.strategy.filteredSet(set2)
        return filteredSet1 & filteredSet2
```

Текст context_union.py:

```
from factory.context_abstract import SetContext


# Контекст работы с операций объединения множеств
class SetUnionContext(SetContext):

    def __init__(self, initialStrategy = None):
        self.strategy = initialStrategy

    def SetStrategy(self, newStrategy):
        SetContext.SetStrategy(self, newStrategy)

    def GetStrategy(self):
        SetContext.GetStrategy(self)

    def ExecuteOperation(self, set1, set2):
        filteredSet1 = self.strategy.filteredSet(set1)
        filteredSet2 = self.strategy.filteredSet(set2)
        return filteredSet1 | filteredSet2
```

Текст strategies.py:

```
from abc import ABC, abstractmethod


# Абстрактный класс для выбора стратегий ("Стратегия")
class SetStrategy(ABC):

    @abstractmethod
    def filteredSet(self, _set):
        pass


# Далее представлены конкретные стратегии фильтрации элементов множества


class NaturalStrategy(SetStrategy):

    def filteredSet(self, _set):
        return set(filter(lambda x: isinstance(x, int) and x > 0, _set))


class DiscreteStrategy(SetStrategy):

    def filteredSet(self, _set):
        return set(filter(lambda x: isinstance(x, int), _set))


class RealStrategy(SetStrategy):

    def filteredSet(self, _set):
        return set(filter(lambda x: isinstance(x, float), _set))


class StringStrategy(SetStrategy):

    def filteredSet(self, _set):
        return set(filter(lambda x: isinstance(x, str) and x[0].upper() == 'H', _set))
```

Текст mocking.py:

```
from strategy.strategies import DiscreteStrategy, NaturalStrategy
# Для примера с подстановкой функции
def mock_setdiff(set1, set2):
    return set1 - set2
class DiffTesting(unittest.TestCase):
    set1 = set([13, -6.1847356, "Hello, mock-object world!", -2,  4])
    set2 = set([4, -1.4142])
    # Демонстрация затрат времени при тестировании без заглушек
    def testWithoutMocks(self):
        context = SetDifferenceContext(DiscreteStrategy())
        diffSet = context.ExecuteOperation(self.set1, self.set2)
        self.assertIn(1, diffSet)

    # Демонстрация работы с mock-объектом при задании возвращаемого значения
функции
    @patch("factory.context_difference.SetDifferenceContext.ExecuteOperation", return_value =
13)
    def testWithPatch_return(self, ExecuteOperation):
```

```
        self.assertEqual(SetDifferenceContext(NaturalStrategy()).ExecuteOperation(self.set1,
self.set2), 13)
    # Демонстрация работы с mock-объектом при задании подменяющей функции
    @patch("factory.context_difference.SetDifferenceContext.ExecuteOperation", side_effect =
mock_setdiff)
    def testWithPatch_side(self, ExecuteOperation):
        self.assertTrue(
            SetDifferenceContext(NaturalStrategy()).ExecuteOperation(self.set1, self.set2) >
                    set([]))
```

Текст tdd.py:

```
from strategy.strategies import NaturalStrategy, DiscreteStrategy, RealStrategy, StringStrategy
from factory.context_union import SetUnionContext
from factory.context_intersection import SetIntersectionContext
from factory.context_difference import SetDifferenceContext
import unittest

class UnionTesting(unittest.TestCase):
    set1 = { 18, 324.7, "123x", -7, 6.13871245 }
    set2 = { "Hello, unittest world!", "hey", -7.62, 1, 18 }

    # Проверка для RealStrategy
    def testFloat(self):
        context = SetUnionContext(RealStrategy())
        unionSet = context.ExecuteOperation(self.set1, self.set2)
        # Строка не должна входить в итоговое множество, а действительное число -
должно
        self.assertNotIn("123x", unionSet)
        self.assertIn(-7.62, unionSet)

    # Проверка для DiscreteStrategy
    def testDiscrete(self):
        context = SetUnionContext(DiscreteStrategy())
        unionSet = context.ExecuteOperation(self.set1, self.set2)
        # Единица должна остаться как часть множества целых чисел
        self.assertIn(1, unionSet)

    # Проверка для NaturalStrategy
    def testNatural(self):
        context = SetUnionContext(NaturalStrategy())
        unionSet = context.ExecuteOperation(self.set1, self.set2)
        # В двух множествах заданы всего два различных натуральных числа
        self.assertEqual(set([1, 18]), unionSet)

    # Проверка для StringStrategy
    def test_str(self):
        context = SetUnionContext(StringStrategy())
        unionSet = context.ExecuteOperation(self.set1, self.set2)
        # Остаются только те строки, которые начинаются с символа "H/h"
        self.assertTrue(unionSet == set(["hey", "Hello, unittest world!"]))


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

Результат работы:

Экранные формы:

```
/usr/bin/python3.7 /snap/pycharm-community/223/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --path /home/foxers/RIP/Lab4/lab4/testing/mo
Testing started at 3:19 ...
Launching unittests with arguments python -m unittest /home/foxers/RIP/Lab4/lab4/testing/mocking.py in /home/foxers/RIP/Lab4/lab4/testing


Ran 3 tests in 0.892s

OK

Process finished with exit code 0
```

```
/usr/bin/python3.7 /snap/pycharm-community/223/plugins/python-ce/helpers/pycharm/_jb_unittest_runner.py --path /home/foxers/RIP/Lab4/lab4/testing/tdd
Testing started at 3:20 ...


Ran 4 tests in 0.002s

OK
Launching unittests with arguments python -m unittest /home/foxers/RIP/Lab4/lab4/testing/tdd.py in /home/foxers/RIP/Lab4/lab4/testing


Process finished with exit code 0
```