



Министерство образования и науки 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)

Лабораторная работа №4 по курсу
«Разработка Интернет-Приложений»

Тема работы: «Шаблоны проектирования и
модульное тестирование в Python»

Выполнил:

Брусов Н.К.

студент группы РТ5-51Б

Проверил: Гапанюк Ю. Е.

10 ноября 2020 г.

ЗАЧТЕНО / НЕ ЗАЧТЕНО _____

(подпись)

г. Москва

Оглавление

Цель выполнения лабораторной работы.....	4
Задание на лабораторную работу.....	4
Ход выполнения лабораторной работы.....	5
1. Реализация шаблонов проектирования.....	5
1.1 Порождающий шаблон проектирования «Фабрика».....	5
1.2 Поведенческий паттерн «Стратегия».....	7
1.3 Структурный паттерн «Мост».....	8
2. Модульное тестирование.....	9
2.1 Тестирование по методике Test-Driven Development (TDD).....	9
2.2 Тестирование по методике Behaviour-Driven Development (BDD).....	10
2.3 Тестирование с использованием Mock-объектов.....	11
Результаты работы.....	13

Цель выполнения лабораторной работы

Изучение шаблонов проектирования, их реализация на языке Python, а также освоение технологии модульного тестирования на примере созданного функционала.

Задание на лабораторную работу

По заданию необходимо реализовать по одному шаблону проектирования из каждой группы: порождающий, структурный, поведенческий. Для каждой реализации шаблона проектирования необходимо написать модульный тест. В модульных тестах необходимо использовать технологии TDD- и BDD-тестирования, а также создания Mock-объектов (заглушек).

Ход выполнения лабораторной работы

Для выполнения лабораторной работы будет использоваться область работы с множествами. Все реализованные шаблоны проектирования будут так или иначе связаны с операциями на множествах.

1. Реализация шаблонов проектирования

1.1 Порождающий шаблон проектирования «Фабрика»

Для начала создаём общий абстрактный класс *SetContext*:

```
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
```

Теперь создаём три класса, реализующих основные операции с множествами – объединение, пересечение и разность.

Класс *SetUnionContext*:

```
from lab4.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
```

Класс *SetIntersectionContext*:

```
from lab4.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
```

Класс *SetDifferenceContext*:

```
from lab4.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 тоск-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
```

1.2 Поведенческий паттерн «Стратегия»

Создаём абстрактный класс *SetStrategy*:

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

    @abstractmethod
    def filteredSet(self, _set):
        pass
```

Теперь создаём четыре конкретных стратегии фильтрации множеств – для отбора натуральных чисел, для отбора целых чисел, для отбора вещественных чисел и для отбора строковых элементов, начинающихся с заглавной буквы «Н».

Класс *NaturalStrategy*:

```
class NaturalStrategy(SetStrategy):

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

Класс *DiscretelStrategy*:

```
class DiscreteStrategy(SetStrategy):

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

Класс *RealStrategy*:

```
class RealStrategy(SetStrategy):

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

Класс *StringStrategy*:

```
class StringStrategy(SetStrategy):

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

1.3 Структурный паттерн «Мост»

Данный паттерн внедрён в уже существующие классы путём внесения стратегии фильтрации множеств в класс операций как поля класса. Это позволяет избежать создания 12 классов (для комбинаций из каждой операции и каждой стратегии) на текущий момент и ещё большего масштабирования в последующие моменты. Вместо этого создаётся по классу под каждую операцию, а стратегию можно внести через соответствующее свойство класса.

2. Модульное тестирование

2.1 Тестирование по методике Test-Driven Development (TDD)

Подключаем для работы встроенную библиотеку unittest и создаём класс для тестирования:

```
from lab4.strategy.strategies import NaturalStrategy, DiscreteStrategy,
RealStrategy, StringStrategy
from lab4.factory.context_union import SetUnionContext
from lab4.factory.context_intersection import SetIntersectionContext
from lab4.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!"]))
```

2.2 Тестирование по методике Behaviour-Driven Development (BDD)

Для данного тестирования используем библиотеку behave. Создаём сценарии тестирования с использованием языка Gherkin:

Feature: Lab4 Behavior Driven Development

Scenario: Test set intersection operation

Given I have sets {1, 'C', -2.1, 36, "1-h", -3} and {"6jsQ", -3, 11, "C"}

When I intersect them with discrete numbers strategy

Then I expect the result to be {-3}

Scenario:

Given I have sets {18, 324.7, "123x", -7, 6.13871245} and {"Hello, unittest world!", "hey", -7.62, 1, 18}

When I intersect them with discrete numbers strategy

Then I expect the result to be {18}

Теперь создаём функции для исполнения сценария:

```
from lab4.factory.context_intersection import SetIntersectionContext
from lab4.strategy.strategies import DiscreteStrategy
from behave import given, when, then
```

```
#Преобразуем строку из сценария теста в множество с элементами разного типа
def SetParser_BDD(str):
```

```
    def isfloat(str):
        try:
            float(str)
            return True
        except ValueError:
            return False
```

```
    def isint(str):
        try:
            int(str)
            return True
        except ValueError:
            return False
```

```
    # Удаляем фигурные скобки и запятые, после чего разбиваем строку на
    множество
    result = set(str[1:-1].replace(',', '').split())
    # Теперь необходимо произвести преобразование элементов обратно к своему
    типу
    newResult = set()
    for element in result:
        if element.startswith("\'") or element.startswith("\'"):
            # Если мы попали в строковый элемент, то убираем лишние кавычки
            newResult.add(element[1:-1])
        elif isint(element):
            # Если мы попали в целое число, то преобразуем его обратно в
            целое число
            newResult.add(int(element))
        elif isfloat(element):
            # Если мы попали в число с плавающей точкой, то преобразуем его
            обратно в число
```

```

        newResult.add(float(element))
    return newResult

@given("I have sets {set1} and {set2}")
def initialSets(context, set1, set2):
    context.set1 = SetParser_BDD(set1)
    context.set2 = SetParser_BDD(set2)

@when("I intersect them with discrete numbers strategy")
def operation(context):
    operationContext = SetIntersectionContext(DiscreteStrategy())
    context.result = operationContext.ExecuteOperation(context.set1,
context.set2)

@then("I expect the result to be {expectedResult}")
def result(context, expectedResult):
    assert SetParser_BDD(expectedResult) == context.result

```

2.3 Тестирование с использованием Mock-объектов

Создаём класс для тестирования:

```

import unittest
from unittest.mock import patch
from lab4.factory.context_difference import SetDifferenceContext
from lab4.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,
"Lab4"])
    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 тоск-объектом при задании возвращаемого значения
    функции

    @patch("lab4.factory.context_difference.SetDifferenceContext.ExecuteOperatio
n", return_value = 13)
    def testWithPatch_return(self, ExecuteOperation):

self.assertEqual(SetDifferenceContext(NaturalStrategy()).ExecuteOperation(se
lf.set1, self.set2), 13)

    # Демонстрация работы с тоск-объектом при задании подменяющей функции

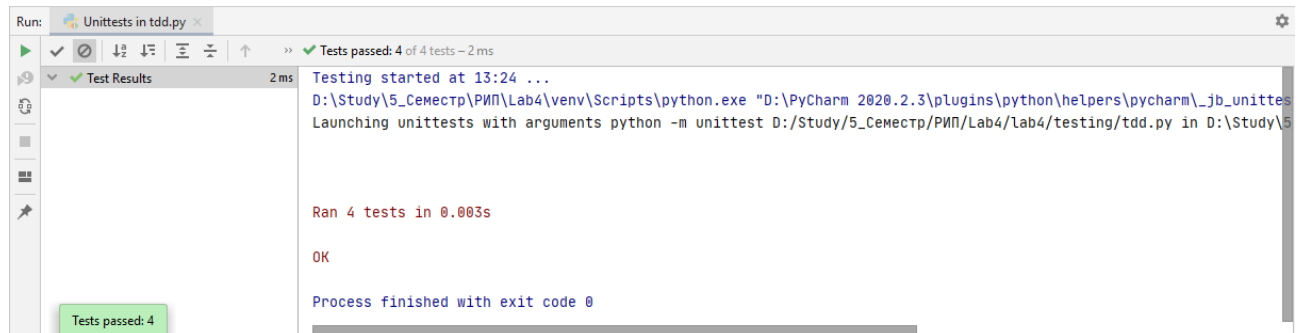
```

```

@patch("lab4.factory.context_difference.SetDifferenceContext.ExecuteOperatio
n", side_effect = mock_setdiff)
    def testWithPatch_side(self, ExecuteOperation):
        self.assertTrue(
SetDifferenceContext(NaturalStrategy()).ExecuteOperation(self.set1,
self.set2) >
                    set([]))

```

Результаты работы



Run: Unittests in tdd.py

Tests passed: 4 of 4 tests - 2 ms

Test Results 2 ms

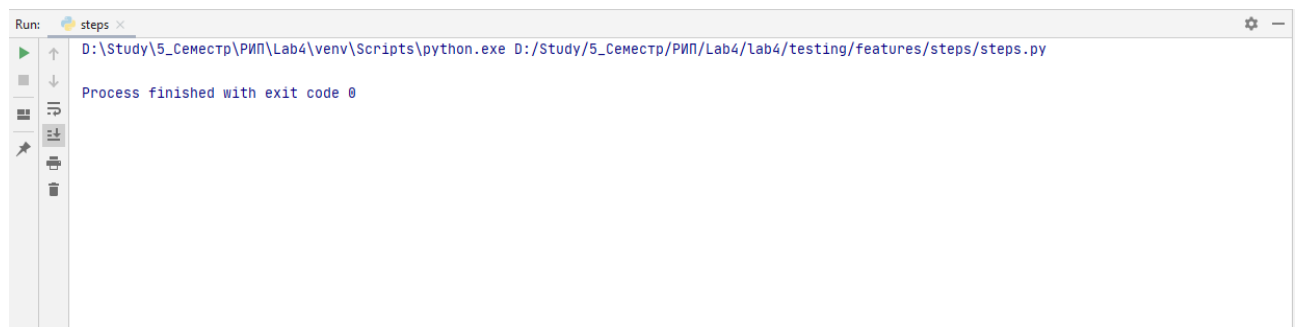
Testing started at 13:24 ...
D:\Study\5_Семестр\ПИП\Lab4\venv\Scripts\python.exe "D:\PyCharm 2020.2.3\plugins\python\helpers\pycharm_jb_unittests.py" D:\Study\5_Семестр\ПИП\Lab4\lab4\testing\tdd.py
Launching unittests with arguments python -m unittest D:/Study/5_Семестр/ПИП/Lab4/testing/tdd.py in D:\Study\5_Семестр\ПИП\Lab4\venv\Scripts\python.exe

Ran 4 tests in 0.003s

OK

Process finished with exit code 0

Tests passed: 4



Run: steps

D:\Study\5_Семестр\ПИП\Lab4\venv\Scripts\python.exe D:/Study/5_Семестр/ПИП/Lab4/lab4/testing/features/steps/steps.py

Process finished with exit code 0



Run: Unittests in mocking.py

Tests passed: 3 of 3 tests - 1 s 591 ms

Test Results 1 s 591 ms

Testing started at 13:24 ...
D:\Study\5_Семестр\ПИП\Lab4\venv\Scripts\python.exe "D:\PyCharm 2020.2.3\plugins\python\helpers\pycharm_jb_unittests.py" D:\Study\5_Семестр\ПИП\Lab4\lab4\testing\mocking.py
Launching unittests with arguments python -m unittest D:/Study/5_Семестр/ПИП/Lab4/testing/mocking.py in D:\Study\5_Семестр\ПИП\Lab4\venv\Scripts\python.exe

Ran 3 tests in 1.594s

OK

Process finished with exit code 0