|  |  |
| --- | --- |
|  | **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:

Забурунов Л. В.

студент группы РТ5-51Б

Проверил: Гапанюк Ю. Е.

10 ноября 2020 г.

ЗАЧТЕНО / НЕ ЗАЧТЕНО\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

г. Москва

Оглавление

[Цель выполнения лабораторной работы 3](#_Toc57548717)

[Задание на лабораторную работу 3](#_Toc57548718)

[Ход выполнения лабораторной работы 4](#_Toc57548719)

[1. Реализация шаблонов проектирования 4](#_Toc57548720)

[1.1 Порождающий шаблон проектирования «Фабрика» 4](#_Toc57548721)

[1.2 Поведенческий паттерн «Стратегия» 6](#_Toc57548722)

[1.3 Структурный паттерн «Мост» 7](#_Toc57548723)

[2. Модульное тестирование 8](#_Toc57548724)

[2.1 Тестирование по методике Test-Driven Development (TDD) 8](#_Toc57548725)

[2.2 Тестирование по методике Behaviour-Driven Development (BDD) 9](#_Toc57548726)

[2.3 Тестирование с использованием Mock-объектов 10](#_Toc57548727)

[Результаты работы 12](#_Toc57548728)

# 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 использовать область работы с множествами. Все реализованные шаблоны проектирования будут так или иначе связаны с операциями на множествах.

# Реализация шаблонов проектирования

# 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 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

# 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 «H».

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))

# 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.

# Модульное тестирование

# 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!"**]))

# 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

# 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 mock-объектом при задании возвращаемого значения функции* @patch(**"lab4.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(**"lab4.factory.context\_difference.SetDifferenceContext.ExecuteOperation"**, side\_effect = mock\_setdiff)  
 **def** testWithPatch\_side(self, ExecuteOperation):  
 self.assertTrue(  
 SetDifferenceContext(NaturalStrategy()).ExecuteOperation(self.set1, self.set2) >  
 set([]))

# Результаты работы





