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 «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Рубежный контроль №2

по дисциплине «Базовые компоненты интернет-технологий»

Выполнила:

студентка группы ИУ5-35Б

Ищенко А.С.

2021 г.

Задание рубежного контроля №2:

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

Задание рубежного контроля № 1:

Вариант Д.

1. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список всех сотрудников, у которых фамилия заканчивается на «ов», и названия их отделов.

2. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список отделов со средней зарплатой сотрудников в каждом отделе, отсортированный по средней зарплате (отдельной функции вычисления среднего значения в Python нет, нужно использовать комбинацию функций вычисления суммы и количества значений).

3. «Отдел» и «Сотрудник» связаны соотношением многие-ко-многим. Выведите список всех отделов, у которых название начинается с буквы «А», и список работающих в них сотрудников.

Вариант предметной области 6.

Дом-Улица

В соответствии с предметной областью, задание было немного изменено:

1. «Улица» и «Дом» связаны соотношением один-ко-многим. Выведите список всех домов, у которых названия заканчивается на «ка», и названия их улиц.

Программа:

Файл main.py:

```
# используется для сортировки
from operator import itemgetter

class House:
    """Дом"""

    def __init__(self, id, name, cost, streetID):
        self.id = id
        self.name = name
        self.cost = cost
        self.streetID = streetID


class Str:
    """Улица"""

    def __init__(self, id, name):
        self.id = id
        self.name = name


class StrHouse:
    """
    'Дома улицы' для реализации
    связи многие-ко-многим
    """

    def __init__(self, streetID, houseID):
        self.streetID = streetID
        self.houseID = houseID


# Улицы
streets = [
    Str(1, 'Пушкинская улица'),
    Str(2, 'Авангардная улица'),
    Str(3, 'Алтайская улица'),

    # для связи многие-ко-многим:
    Str(11, 'Авиационная улица'),
    Str(22, 'Новорязанская улица'),
    Str(33, 'Парковая улица'),
]

# Сотрудники
houses = [
    House(1, 'Малосемейка', 7000000, 1),
    House(2, 'Сталинка', 20000000, 2),
    House(3, 'Хрущевка', 12000000, 2),
    House(4, 'Брежневка', 18000000, 3),
    House(5, 'Студия', 10000000, 3),
]
```

```
strsHouses = [
    StrHouse(1, 1),
    StrHouse(2, 2),
    StrHouse(2, 3),
    StrHouse(3, 4),
    StrHouse(3, 5),

    StrHouse(11, 1),
    StrHouse(22, 2),
    StrHouse(22, 3),
    StrHouse(33, 4),
    StrHouse(33, 5),

]

# Декоратор (сделала по личной инициативе)
def print_result(func):
    def wrapper(*kwargs):
        #составляем имя задания (буква D + номер задания (для всех
функций начинается с 4 места после слова task)
        name = "D" + func.__name__[4:]
        print("Задание " + name)
        result = func(*kwargs)
        print(result)
        return result
    return wrapper

@print_result
def task1(one_to_many):
    res1 = list(filter(lambda x: x[0].endswith("ка"), one_to_many))
    return res1

@print_result
def task2(one_to_many):
    res2unsorted = []
    # Перебираем все улицы
    for s in streets:
        # Список домов на улице
        houses1 = list(filter(lambda i: i[2] == s.name, one_to_many))
        # Если на улице есть дома
        if len(houses1) > 0:
            # Все цены домов на улице
            allCosts = [sal for _, sal, _ in houses1]
            # Средняя цена дома на улице
            averageCosts = round(sum(allCosts) / len(allCosts), 1)
            res2unsorted.append((s.name, averageCosts))

    # Сортировка по средней стоимости
    res2 = sorted(res2unsorted, key=itemgetter(1), reverse=True)
    return res2

@print_result
def task3(many_to_many):
    res3 = {}
    # Цикл по всем улицам
    for s in streets:
        if s.name.startswith("A"):
```

```
            # Список домов на улице
            houses1 = list(filter(lambda i: i[2] == s.name,
many_to_many))
            # Только имя дома
            housesNames = [x for x, _, _ in houses1]
            # Добавляем результат в словарь
            # ключ - улица, значение - список названий домов
            res3[s.name] = housesNames

    return res3


def relations_one_to_many():

    # Соединение данных один-ко-многим с помощью кортежа
    one_to_many = [(h.name, h.cost, s.name)
                   for s in streets
                   for h in houses
                   if h.streetID == s.id]
    return one_to_many

def relations_many_to_many():
    # Соединение данных многие-ко-многим с помощью кортежа
    many_to_many_temp = [(s.name, sh.streetID, sh.houseID)
                         for s in streets
                         for sh in strsHouses
                         if s.id == sh.streetID]

    many_to_many = [(h.name, h.cost, streetName)
                    for streetName, streetID, houseID in
many_to_many_temp
                    for h in houses if h.id == houseID]
    return  many_to_many


if __name__ == '__main__':
    one_to_many = relations_one_to_many()
    many_to_many = relations_many_to_many()

    task1(one_to_many)
    task2(one_to_many)
    task3(many_to_many)
```

Файл tdd.py:

```
import unittest
import main


class TestStreetsAndHouses(unittest.TestCase):

    def test_task1(self):
        relations = main.relations_one_to_many()
        expected_result = [('Малосемейка', 7000000, 'Пушкинская
улица'), ('Сталинка', 20000000, 'Авангардная улица'),
                           ('Хрущевка', 12000000, 'Авангардная улица'),
('Брежневка', 18000000, 'Алтайская улица')]
        self.assertEqual(main.task1(relations), expected_result)
    def test_task2(self):
        relations = main.relations_one_to_many()
        expected_result = [('Авангардная улица', 16000000.0),
('Алтайская улица', 14000000.0), ('Пушкинская улица', 7000000.0)]
        self.assertEqual(main.task2(relations), expected_result)
    def test_task3(self):
        relations = main.relations_many_to_many()
        expected_result = {'Авангардная улица': ['Сталинка',
'Хрущевка'],
                           'Алтайская улица': ['Брежневка', 'Студия'],
'Авиационная улица': ['Малосемейка']}
        self.assertEqual(main.task3(relations), expected_result)

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

Результаты выполнения программы:

Файл main.py:

```
Задание D1
[('Малосемейка', 7000000, 'Пушкинская улица'), ('Сталинка', 20000000, 'Авангардная улица'), ('Хрущевка', 12000000, 'Авангардная улица'), ('Брежневка', 18000000, 'Алтайская улица')]
Задание D2
[('Авангардная улица', 16000000.0), ('Алтайская улица', 14000000.0), ('Пушкинская улица', 7000000.0)]
Задание D3
{'Авангардная улица': ['Сталинка', 'Хрущевка'], 'Алтайская улица': ['Брежневка', 'Студия'], 'Авиационная улица': ['Малосемейка']}

Process finished with exit code 0
```

Файл tdd.py:

```
Testing started at 14:34 ...
Launching unittests with arguments python -m unittest


Process finished with exit code 0


Ran 3 tests in 0.003s

OK
```