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»

Вариант №3

Выполнила:
студентка группы ИУ5-33Б
Валова С. В.

Проверил:
Преподаватель кафедры ИУ-5
Гапанюк Ю. Е.

2022 г.

Условие

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).

Условия РК №1:

1. «Водитель» и «Автопарк» связаны соотношением один-ко-многим. Выведите список всех водителей, у которых фамилия начинается с буквы «А», и названия их автопарков.

2. «Водитель» и «Автопарк» связаны соотношением один-ко-многим. Выведите список автопарков с минимальной зарплатой водителей в каждом автопарке, отсортированный по минимальной зарплате.

3. «Водитель» и «Автопарк» связаны соотношением многие-ко-многим. Выведите список всех связанных водителей и автопарков, отсортированный по водителям, сортировка по автопаркам произвольная.

Текст программы

Файл classes.py (

```
class Driver:
    def __init__(self, id, fio, pay, id_autopark):
        self.id = id
        self.fio = fio
        self.pay = pay
        self.IdAutopark = id_autopark

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

class Ap_to_Dr:
    def __init__(self, id_Ap, id_Dr):
        self.id_Ap = id_Ap
        self.id_Dr = id_Dr

Drivers = [
    Driver(1, 'Артаев', 20000, 1),
    Driver(2, 'Алешин', 22000, 2),
    Driver(3, 'Светличко', 20000, 1),
    Driver(4, 'Кунцевский', 25000, 3),
    Driver(5, 'Аверьянов', 26000, 3),
    Driver(6, 'Приходько', 21000, 1)
]

Autoparks = [
    Autopark(1, 'М-Такси'),
    Autopark(2, 'БасЭлитСервис'),
    Autopark(7, 'Авангард'),
    Autopark(3, 'Яндекс-такси'),
    Autopark(12, 'ТК Повозкин')
]

Aps_Drs = [
```

```
    Ap_to_Dr(1, 1),
    Ap_to_Dr(2, 2),
    Ap_to_Dr(1, 3),
    Ap_to_Dr(3, 4),
    Ap_to_Dr(3, 5),
    Ap_to_Dr(4, 6),
    Ap_to_Dr(1, 7)
]
```

Файл connections.py (реализованы связи один-ко-многим, многие-ко-многим)

```
from classes import *

one_to_many = [(i.fio, i.pay, j.name)
               for i in Drivers
               for j in Autoparks
               if i.IdAutopark == j.id]

many_to_many_temp = [(a.name, n.id_Ap, n.id_Dr)
                     for a in Autoparks
                     for n in Aps_Drs
                     if a.id == n.id_Ap]

many_to_many = [(i.fio, i.pay, Ap_name)
                for Ap_name, id_Ap, id_Dr in many_to_many_temp
                for i in Drivers
                if i.id == id_Dr]
```

Файл task1.py

```
def a_surname_drivers(one_to_many):
    res = list(filter(lambda i: i[0][0] == "A", one_to_many))
    return res
```

Файл task2.py

```
def parks_with_min_drivers_pay(one_to_many):
    list_of_parks = []
    res = []
    for i in one_to_many:
        if i[2] not in list_of_parks:
            list_of_parks.append(i[2])
    for j in list_of_parks:
        min = 100000000
        for i in one_to_many:
            if j == i[2] and i[1] < min:
                min = i[1]
        res.append((j, min))
    return res
```

Файл task3.py

```
from operator import itemgetter

def drivers_sort(one_to_many):
    res = sorted(one_to_many, key=itemgetter(0))
    return res
```

Файл с тестом TDD-tests.py

```
import unittest
from connections import *
from task1 import a_surname_drivers
```

```
from task2 import parks_with_min_drivers_pay
from task3 import drivers_sort


class Testing(unittest.TestCase):
    def test_1(self):
        expected_res = [('Артаев', 20000, 'М-Такси'), ('Алешин', 22000,
'БасЭлитСервис'),
                        ('Аверьянов', 26000, 'Яндекс-такси')]
        self.assertEqual(a_surname_drivers(one_to_many), expected_res)

    def test_2(self):
        expected_res = [('М-Такси', 20000), ('БасЭлитСервис', 22000),
('Яндекс-такси', 25000)]
        self.assertEqual(parks_with_min_drivers_pay(one_to_many),
expected_res)

    def test_3(self):
        expected_res = [('Аверьянов', 26000, 'Яндекс-такси'), ('Алешин',
22000, 'БасЭлитСервис'),
                        ('Артаев', 20000, 'М-Такси'),
                        ('Кунцевский', 25000, 'Яндекс-такси'),
                        ('Приходько', 21000, 'М-Такси'),
                        ('Светличко', 20000, 'М-Такси')]
        self.assertEqual(drivers_sort(one_to_many), expected_res)
```

Результат выполнения