**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 №2по 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 сlasses.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] == "А", 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

