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 «Системы обработки информации и управления»

Рубежный контроль №2

Преподаватель
Гапанюк Ю.Е.
04.12.2022

Студент группы ИУ5-34Б
Лавренов М.А.
04.12.2022

Москва

2022 г.

Полученное задание:

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).

Вариант Д 9

9	ОС	ПК

1. «ОС» и «ПК» связаны соотношением один-ко-многим. Выведите список всех ПК, у которых модель начинается с буквы «А», и список установленных на нем ОС.
2. «ОС» и «ПК» связаны соотношением один-ко-многим. Выведите список ПК с максимальным требованием к оперативной памяти ОС, отсортированный по максимальному требуемому объему памяти.
3. «ОС» и «ПК» связаны соотношением многие-ко-многим. Выведите список всех связанных ОС и ПК, отсортированный по моделям ПК, сортировка по ОС произвольная.

Текст программы

Программа разделена на несколько файлов.

1) Файл main.py содержит классы для реализации задания:

```
from operator import itemgetter
class Director:
    """Дирижер"""
    def __init__(self, id, name, salary, orch_id):
        self.id = id
        self.name = name
        self.salary = salary
        self.orch_id = orch_id
class Orchestra:
    """Оркестр"""
    def __init__(self, id, title):
        self.id = id
        self.title = title
class DirectorOrchestra:
    """
     'Дирижеры оркестров' для реализации
     связи многие-ко-многим
     """

    def __init__(self, dir_id, orch_id):
        self.dir_id = dir_id
```

```
        self.orch_id = orch_id


# Дирижеры
directors = [
    Director(1, 'Бан', 150000, 1),
    Director(2, 'Вагнер', 2800, 2),
    Director(3, 'Деймур', 100000, 2),
    Director(4, 'Ким Чен Ын', 86400, 3),
    Director(5, 'Поляков', 44444, 4)
]
# Оркестры
orchestras = [
    Orchestra(1, 'Банановая республика'),
    Orchestra(2, 'AGSPD'),
    Orchestra(3, 'BTS'),
    Orchestra(4, 'Дрилл Авиация')
]
directors_orchestras = [
    DirectorOrchestra(1, 1),
    DirectorOrchestra(2, 2),
    DirectorOrchestra(3, 3),
    DirectorOrchestra(4, 4),
    DirectorOrchestra(5, 1),
    DirectorOrchestra(5, 1),
    DirectorOrchestra(2, 3),
    DirectorOrchestra(3, 2),
    DirectorOrchestra(4, 1),
    DirectorOrchestra(5, 4)
]

one_to_many = [(dir.name, dir.salary, orch.title)
                   for orch in orchestras
                   for dir in directors
                   if dir.orch_id == orch.id]
# Соединение данных многие-ко-многим
many_to_many_temp = [(orch.title, dir.orch_id, dir.dir_id)
                     for orch in orchestras
                     for dir in directors_orchestras
                     if orch.id == dir.orch_id]
many_to_many = [(dir.name, dir.salary, orchestra_name)
                for orchestra_name, orch_id, dir_id in many_to_many_temp
                for dir in directors if dir.id == dir_id]

def task1(one_to_many):
    res = {}
    for orch in orchestras:
        if orch.title[0] == 'A':
            d_emps = list(filter(lambda i: i[2] == orch.title,
one_to_many))
    d_emps_names = [x for x, _, _ in d_emps]
    res[orch.title] = d_emps_names
    return res

def task2(one_to_many):
    res2_unsorted = []
    for orch in orchestras:
        orch_Directors = list(filter(lambda i: i[2] == orch.title,
                                     one_to_many))
    if len(orch_Directors) > 0:
        orch_sals = [sal for _, sal, _ in orch_Directors]
    orch_sals_sum = max(orch_sals)
    res2_unsorted.append((orch.title, orch_sals_sum))
```

```
    res2 = sorted(res2_unsorted, key=itemgetter(1), reverse=True)
    return res2

def task3(many_to_many):
    res3 = sorted(many_to_many, key=itemgetter(2))
    return res3




# print('\nЗадание Г1')
# print(task1(one_to_many))
# print('\nЗадание Г2')
# print(task2(one_to_many))
# print('\nЗадание Г3')
# for i in task3(many_to_many):
#     print(i)
```

2) Файл testing.py реализует тестирование программы:

```
import unittest
import sys,os

sys.path.append(os.getcwd()) #current working directory
from main import *
from main import task1, task2, task3
from main import one_to_many, many_to_many


class TestTask1(unittest.TestCase):
    def test_task_1(self):
        res = task1(one_to_many)
        expected = {'Дрилл Авиация': ['Вагнер', 'Деймур']}
        self.assertEqual(res, expected)

class TestTask2(unittest.TestCase):
    def test_task_2(self):
        res = task2(one_to_many)
        expected = [('Дрилл Авиация', 44444)]
        self.assertEqual(res, expected)


class TestTask3(unittest.TestCase):
    def test_task_3(self):
        res = task3(many_to_many)
        expected = [('Вагнер', 2800, 'AGSPD'),
                    ('Деймур', 100000, 'AGSPD'),
                    ('Деймур', 100000, 'BTS'),
                    ('Вагнер', 2800, 'BTS'),
                    ('Бан', 150000, 'Банановая республика'),
                    ('Поляков', 44444, 'Банановая республика'),
                    ('Поляков', 44444, 'Банановая республика'),
                    ('Ким Чен Ын', 86400, 'Банановая республика'),
                    ('Ким Чен Ын', 86400, 'Дрилл Авиация'),
                    ('Поляков', 44444, 'Дрилл Авиация')]
        self.assertEqual(res, expected)



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

Результат работы программы