Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

Вариант E18

Выполнил:
студент группы РТ5-31Б:
Савельева В. О.

Подпись и дата:

Проверил:
преподаватель кафедры ИУ5
Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 г.

Постановка задачи

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).

Текст программы

```
import unittest
class Employee:  # Музыкальное произведение
    def __init__(self, ID: int, surname: str, salary: int, IDu: int):
        self.ID = ID
        self.surname = surname  # Название композиции
        self.salary = salary  # Продолжительность произведения в секундах
        self.IDu = IDu

class Unit:  # Оркестр
    def __init__(self, ID: int, name: str):
        self.ID = ID
        self.name = name

class EmpUn:
    def __init__(self, mIDe: int, mIDu: int):
        self.mIDe = mIDe
        self.mIDu = mIDu

def initialize_orchestras():
    return [
        Unit(1, "Симфонический оркестр Мариинского театра"),
        Unit(2, "Государственный академический симфонический оркестр России имени
Е. Ф. Светланова"),
        Unit(3, "Московский государственный академический симфонический оркестр")
    ]

def initialize_music():
    return [
        Employee(1, "К Элизе", 164, 1),
        Employee(2, "Лунная соната", 307, 1),
        Employee(3, "Пришла весна", 200, 2),
        Employee(4, "Турецкий марш", 208, 2),
        Employee(5, "Лунный свет", 320, 3),
        Employee(6, "Шторм", 171, 3)
    ]

def initialize_emp_un():
    return [
```

```
        EmpUn(1, 1),
        EmpUn(1, 3),
        EmpUn(2, 1),
        EmpUn(2, 2),
        EmpUn(3, 2),
        EmpUn(3, 3),
        EmpUn(4, 2),
        EmpUn(5, 3),
        EmpUn(6, 3)
    ]

def one_to_many(music_list, orchestra_list):
    return [(o.name, m.surname, m.salary)
            for m in music_list
            for o in orchestra_list
            if m.IDu == o.ID]

def many_to_many_temp(orchestra_list, emp_un_list):
    return [(o.name, mo.mIDu, mo.mIDe)
            for o in orchestra_list
            for mo in emp_un_list
            if o.ID == mo.mIDu]

def many_to_many(music_list, many_to_many_temp_list):
    return [(m.surname, m.salary, oname)
            for oname, oID, mID in many_to_many_temp_list
            for m in music_list if m.ID == mID]

def e1(one_to_many_list):
    return sorted(filter(lambda x: "оркестр" in x[0], one_to_many_list))

def e2(one_to_many_list):
    sorted_list = sorted(one_to_many_list)
    res = []
    temp_name = ""
    sum_duration = count = 0
    for g, i, f in sorted_list:
        if g != temp_name:
            if count != 0:
                average_duration = round(sum_duration / count, 2)
                res.append([average_duration, temp_name])
            temp_name = g
            sum_duration = f
            count = 1
        else:
            sum_duration += f
            count += 1
    if count != 0:
        average_duration = round(sum_duration / count, 2)
        res.append([average_duration, temp_name])
    return sorted(res)
```

```
def e3(many_to_many_list):
    return sorted(filter(lambda x: x[0][0] == "Л", many_to_many_list))


class Tests(unittest.TestCase):
    def setUp(self):
        self.orchestra = initialize_orchestras()
        self.music = initialize_music()
        self.emp_un = initialize_emp_un()
        self.one_to_many_list = one_to_many(self.music, self.orchestra)
        self.many_to_many_temp_list = many_to_many_temp(self.orchestra,
self.emp_un)
        self.many_to_many_list_result = many_to_many(self.music,
self.many_to_many_temp_list)

    def test_e1(self):
        result = e1(self.one_to_many_list)
        expectation = [("Государственный академический симфонический оркестр
России имени Е. Ф. Светланова", "Пришла весна", 200),
                       ("Государственный академический симфонический оркестр
России имени Е. Ф. Светланова", "Турецкий марш", 208),
                       ("Московский государственный академический симфонический
оркестр", "Лунный свет", 320),
                       ("Московский государственный академический симфонический
оркестр", "Шторм", 171),
                       ("Симфонический оркестр Мариинского театра", "К Элизе",
164),
                       ("Симфонический оркестр Мариинского театра", "Лунная
соната", 307)
                       ]
        self.assertEqual(result, expectation)
    def test_e2(self):
        result = e2(self.one_to_many_list)
        expectation = [[round((200+208) / 2, 2), "Государственный академический
симфонический оркестр России имени Е. Ф. Светланова"],
                       [round((164+307) / 2, 2), "Симфонический оркестр
Мариинского театра"],
                       [round((320+171) / 2, 2), "Московский государственный
академический симфонический оркестр"]
                       ]
        self.assertEqual(result, expectation)
    def test_e3(self):
        result = e3(self.many_to_many_list_result)
        expectation = [("Лунная соната", 307, "Государственный академический
симфонический оркестр России имени Е. Ф. Светланова"),
                       ("Лунная соната", 307, "Симфонический оркестр Мариинского
театра"),
                       ("Лунный свет", 320, "Московский государственный
академический симфонический оркестр")
                       ]
```

```
        self.assertEqual(result, expectation)

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

Результат

```
Ran 3 tests in 0.000s

OK
PS C:\pclp\rk2>
```