Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по РК №2

Вариант В, номер 5

Выполнила:
студентка группы ИУ5-33Б
Буйдина Кристина

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.

Москва, 2023 г.

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).

Код программы:

main.py

```
from operator import itemgetter

class Musician:
    """Музыкант"""

    def __init__(self, id, fio, sal, orch_id):
        if id < 0:
            raise ValueError("ID музыканта не может быть отрицательным")
        self.id = id
        self.fio = fio
        if sal < 0:
            raise ValueError("Зарплата не может быть отрицательной")
        self.sal = sal
        if orch_id < 0:
            raise ValueError("ID орекстра не может быть отрицательным")
        self.orch_id = orch_id


class Orchestra:
    """Оркестр"""

    def __init__(self, id, name):
        if id < 0:
            raise ValueError("ID орекстра не может быть отрицательным")
        self.id = id
        self.name = name


class MusOrch:
    """
    'Музыканты Оркестра' для реализации
    связи многие-ко-многим
    """

    def __init__(self, orch_id, mus_id):
        if orch_id < 0:
            raise ValueError("ID орекстра не может быть отрицательным")
        if mus_id < 0:
            raise ValueError("ID музыканта не может быть отрицательным")
        self.orch_id = orch_id
        self.mus_id = mus_id


def task1(one_to_many):
    return sorted([(fio, sal, name) for fio, sal, name in one_to_many if
fio.startswith('A')], key=itemgetter(2))


def task2(one_to_many, orchestras):
```

```
    res_12_unsorted = []
    for d in orchestras:
        d_emps = list(filter(lambda i: i[2] == d.name, one_to_many))
        if len(d_emps) > 0:
            d_sals = [sal for _, sal, _ in d_emps]
            d_sals_min = min(d_sals)
            res_12_unsorted.append((d.name, d_sals_min))
    return sorted(res_12_unsorted, key=itemgetter(1),)

def task3(many_to_many, orchestras):
    res_13 = {}
    sorted_many_to_many = sorted(many_to_many, key=itemgetter(0))
    for d in orchestras:
        d_emps = list(filter(lambda i: i[2] == d.name, many_to_many))
        d_emps_names = [x for x, _, _ in d_emps]
        res_13[d.name] = d_emps_names
    return res_13


def main():
    orchestras = [
        Orchestra(1, 'деревянные духовые'),
        Orchestra(2, 'медные духовые'),
        Orchestra(3, 'струнные смычковые'),
        Orchestra(4, 'ударные'),
        Orchestra(11, 'деревянные духовые (другое)'),
        Orchestra(22, 'медные духовые (другое)'),
        Orchestra(33, 'струнные смычковые (другое)'),
        Orchestra(44, 'ударные (другое)'),
    ]

    musicians = [
        Musician(1, 'Артамонов', 25000, 1),
        Musician(2, 'Петров', 35000, 2),
        Musician(3, 'Иваненко', 45000, 3),
        Musician(4, 'Иванов', 35000, 3),
        Musician(5, 'Сушкин', 25000, 4),
    ]

    musicians_orchestras = [
        MusOrch(1, 1),
        MusOrch(2, 2),
        MusOrch(3, 3),
        MusOrch(3, 4),
        MusOrch(4, 5),
        MusOrch(11, 1),
        MusOrch(22, 2),
        MusOrch(33, 3),
        MusOrch(33, 4),
        MusOrch(44, 5),
    ]

    one_to_many = [(e.fio, e.sal, d.name)
                   for d in orchestras
                   for e in musicians
                   if e.orch_id == d.id]

    many_to_many_temp = [(d.name, ed.orch_id, ed.mus_id)
                         for d in orchestras
                         for ed in musicians_orchestras
                         if d.id == ed.orch_id]

    many_to_many = [(e.fio, e.sal, dep_name)
                    for dep_name, dep_id, emp_id in many_to_many_temp
```

```
                    for e in musicians if e.id == emp_id]

    print('Задание 1')
    print(task1(one_to_many))

    print('\nЗадание 2')
    print(task2(one_to_many, orchestras))

    print('\nЗадание 3')
    print(task3(many_to_many, orchestras))


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

tests.py

```
import unittest
from main import *


class TestMusician(unittest.TestCase):
    """Тестовый класс для класса Musician"""

    def test_Musician(self):
        """Тестирование конструктора класса Musician"""
        m = Musician(1, 'F I O', 0, 2)
        self.assertEqual(m.id, 1)  # Проверка id
        self.assertEqual(m.fio, 'F I O')  # Проверка ФИО
        self.assertEqual(m.sal, 0)  # Проверка зарплаты
        self.assertEqual(m.orch_id, 2)  # Проверка id оркестра

    def test_negative_salary(self):
        """Тестирование конструктора класса Musician с отрицательной
зарплатой"""
        with self.assertRaises(ValueError) as e:
            m = Musician(1, 'Иванов', -1000, 1)
        self.assertEqual(str(e.exception), "Зарплата не может быть
отрицательной")

    def test_negative_id(self):
        """Тестирование конструктора класса Musician с отрицательным id"""
        with self.assertRaises(ValueError) as e:
            m = Musician(-1, 'Иванов', 1000, 1)
        self.assertEqual(str(e.exception), "ID музыканта не может быть
отрицательным")

    def test_negative_orch_id(self):
        """Тестирование конструктора класса Musician с отрицательным id
оркестра"""
        with self.assertRaises(ValueError) as e:
            m = Musician(1, 'Иванов', 1000, -1)
        self.assertEqual(str(e.exception), "ID орекстра не может быть
отрицательным")


class TestOrchestra(unittest.TestCase):
    """Тестовый класс для класса Orchestra"""

    def test_Orchestra(self):
        """Тестирование конструктора класса Orchestra"""
        o = Orchestra(1, 'Bublik')
```

```
        self.assertEqual(o.id, 1)  # Проверка id
        self.assertEqual(o.name, 'Bublik')  # Проверка названия

    def test_negative_id(self):
        """Тестирование конструктора класса Orchestra с отрицательным id"""
        with self.assertRaises(ValueError) as e:
            o = Orchestra(-1, 'Струнные')
        self.assertEqual(str(e.exception), "ID оркестра не может быть
отрицательным")


class TestMusOrch(unittest.TestCase):
    """Тестовый класс для класса MusOrch"""

    def test_MusOrch(self):
        """Тестирование конструктора класса MusOrch"""
        mo = MusOrch(3, 5)
        self.assertEqual(mo.orch_id, 3)  # Проверка id оркестра
        self.assertEqual(mo.mus_id, 5)  # Проверка id музыканта

    def test_negative_orch_id(self):
        """Тестирование конструктора класса MusOrch с отрицательным id
оркестра"""
        with self.assertRaises(ValueError) as e:
            mo = MusOrch(-1, 1)
        self.assertEqual(str(e.exception), "ID оркестра не может быть
отрицательным")

    def test_negative_mus_id(self):
        """Тестирование конструктора класса MusOrch с отрицательным id
музыканта"""
        with self.assertRaises(ValueError) as e:
            mo = MusOrch(1, -1)
        self.assertEqual(str(e.exception), "ID музыканта не может быть
отрицательным")


class TestTasks(unittest.TestCase):
    """Тестовый класс для функций task1, task2 и task3"""

    def test_1(self):
        """Тестирование функции task1"""
        right = [('Артамонов', 25000, 'деревянные духовые')]
        self.assertEqual('foo'.upper(), 'FOO')

    def test_2(self):
        """Тестирование функции task2"""
        right = [('деревянные духовые', 25000), ('ударные', 25000), ('медные
духовые', 35000),
                 ('струнные смычковые', 35000)]
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_3(self):
        """Тестирование функции task3"""
        right = {'деревянные духовые': ['Артамонов'],
                 'медные духовые': ['Петров'],
                 'струнные смычковые': ['Иваненко', 'Иванов'],
                 'ударные': ['Сушкин'],
                 'деревянные духовые (другое)': ['Артамонов'],
                 'медные духовые (другое)': ['Петров'],
                 'струнные смычковые (другое)': ['Иваненко', 'Иванов'],
                 'ударные (другое)': ['Сушкин']}
```

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

Вывод:


```
Launching unittests with arguments pythor


Ran 12 tests in 0.003s

OK

Process finished with exit code 0
```