Московский государственный технический университет им. Н.Э. Баумана

Факультет информатика и системы управления
Кафедра системы обработки информации и управления

Курс «Парадигмы и конструкции языков программирования» Отчет по рубежному контролю №2

Выполнил:
студент группы ИУ5-32Б:
Щепнов Ф.В.
Подпись и дата: 20.12.25

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2025 г.

Постановка задачи

В рамках рубежного контроля №2 по курсу «Парадигмы и конструкции языков программирования» требовалось разработать программу на языке Python.

Цель работы:

1. Провести 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

Текст программы

main.py

```python
from operator import itemgetter


class Composition:
    """Музыкальное произведение"""

    def __init__(self, id, title, duration, orchestra_id):
        self.id = id
        self.title = title
        self.duration = duration
        self.orchestra_id = orchestra_id


class Orchestra:
    """Оркестр"""

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


class CompositionOrchestra:
    """Связь многие-ко-многим между произведениями и оркестрами"""

    def __init__(self, orchestra_id, composition_id):
        self.orchestra_id = orchestra_id
        self.composition_id = composition_id


# Данные (остаются глобальными для запуска main, но передаются в функции)
orchestras = [
    Orchestra(1, 'Симфонический оркестр'),
    Orchestra(2, 'Камерный оркестр'),
    Orchestra(3, 'Джазовый оркестр'),
    Orchestra(4, 'Народный оркестр'),
]

compositions = [
    Composition(1, 'Симфония №5', 20, 1),
    Composition(2, 'Концерт для скрипки', 15, 2),
    Composition(3, 'Рапсодия в стиле блюз', 10, 3),
    Composition(4, 'Симфония №9', 25, 1),
    Composition(5, 'Увертюра 1812', 15, 1),
    Composition(6, 'Симфониетта', 12, 2),
]

compositions_orchestras = [
    CompositionOrchestra(1, 1),
    CompositionOrchestra(1, 4),
    CompositionOrchestra(1, 5),
    CompositionOrchestra(2, 2),
    CompositionOrchestra(2, 6),
    CompositionOrchestra(3, 3),
    CompositionOrchestra(4, 1),
    CompositionOrchestra(4, 3),
]


def get_one_to_many(orchestras, compositions):
    """Формирует список связей один-ко-многим"""
    return [(c.title, c.duration, o.name)
            for o in orchestras
            for c in compositions
```

```
            if c.orchestra_id == o.id]


def get_many_to_many(orchestras, compositions, compositions_orchestras):
    """Формирует список связей многие-ко-многим"""
    many_to_many_temp = [(o.name, co.orchestra_id, co.composition_id)
                         for o in orchestras
                         for co in compositions_orchestras
                         if o.id == co.orchestra_id]

    return [(c.title, c.duration, orchestra_name)
            for orchestra_name, orchestra_id, composition_id in many_to_many_temp
            for c in compositions if c.id == composition_id]


def task_1(one_to_many):
    """
    Задание Б1: Вывести список всех связанных произведений и оркестров,
    отсортированный по названию произведения.
    """
    return sorted(one_to_many, key=itemgetter(0))


def task_2(orchestras, one_to_many):
    """
    Задание Б2: Вывести список оркестров с количеством произведений в каждом,
    отсортированный по количеству произведений.
    """
    res_2_unsorted = []
    for o in orchestras:
        # Фильтруем one_to_many по имени текущего оркестра
        o_compositions = list(filter(lambda i: i[2] == o.name, one_to_many))
        o_compositions_count = len(o_compositions)
        res_2_unsorted.append((o.name, o_compositions_count))

    return sorted(res_2_unsorted, key=itemgetter(1), reverse=True)


def task_3(many_to_many, ending):
    """
    Задание Б3: Вывести список всех оркестров, у которых название произведения
    заканчивается на заданную подстроку (например, 'ов').
    """
    res_3 = []
    for title, duration, orchestra in many_to_many:
        if title.endswith(ending):
            res_3.append((title, orchestra))

    return list(set(res_3))


def main():
    """Основная функция"""
    one_to_many = get_one_to_many(orchestras, compositions)
    many_to_many = get_many_to_many(orchestras, compositions,
compositions_orchestras)

    print('Задание Б1')
    res_1 = task_1(one_to_many)
    [print(f"{title} - {orchestra}") for title, duration, orchestra in res_1]

    print('\nЗадание Б2')
    res_2 = task_2(orchestras, one_to_many)
    [print(f"{orchestra}: {count} произведений") for orchestra, count in res_2]

    print('\nЗадание Б3')
    # В оригинальном задании ищется окончание "ов"
    res_3 = task_3(many_to_many, 'ов')
```

```
    [print(f"{title} - {orchestra}") for title, orchestra in res_3]


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

test_rk2.py

```
import unittest
from main import (
    Composition, Orchestra, CompositionOrchestra,
    get_one_to_many, get_many_to_many,
    task_1, task_2, task_3
)


class TestRK2(unittest.TestCase):
    def setUp(self):
        """Инициализация тестовых данных перед каждым тестом"""
        self.orchestras = [
            Orchestra(1, 'Оркестр А'),
            Orchestra(2, 'Оркестр Б'),
            Orchestra(3, 'Оркестр В'),
        ]

        self.compositions = [
            Composition(1, 'Альфа', 10, 1),
            Composition(2, 'Бета', 20, 1),
            Composition(3, 'Гамма', 15, 2),
            Composition(4, 'Марш Петров', 5, 3),  # Для теста Б3 (оканчивается на
"ов")
        ]

        self.compositions_orchestras = [
            CompositionOrchestra(1, 1),  # Оркестр А - Альфа
            CompositionOrchestra(1, 2),  # Оркестр А - Бета
            CompositionOrchestra(2, 3),  # Оркестр Б - Гамма
            CompositionOrchestra(3, 4),  # Оркестр В - Марш Петров
        ]

        self.one_to_many = get_one_to_many(self.orchestras, self.compositions)
        self.many_to_many = get_many_to_many(self.orchestras, self.compositions,
self.compositions_orchestras)

    def test_task_1(self):
        """Тест задания Б1: сортировка по названию"""
        result = task_1(self.one_to_many)

        # Ожидаем сортировку: Альфа, Бета, Гамма, Марш Петров
        expected_titles = ['Альфа', 'Бета', 'Гамма', 'Марш Петров']
        actual_titles = [item[0] for item in result]

        self.assertEqual(actual_titles, expected_titles)

    def test_task_2(self):
        """Тест задания Б2: сортировка по количеству произведений"""
        result = task_2(self.orchestras, self.one_to_many)

        # Оркестр А: 2 произведения
        # Оркестр Б: 1 произведение
        # Оркестр В: 1 произведение

        # Проверяем, что первый элемент имеет наибольшее количество
        self.assertEqual(result[0][0], 'Оркестр А')
        self.assertEqual(result[0][1], 2)

    def test_task_3(self):
```

```python
        """Тест задания Б3: фильтрация по окончанию"""
        # Ищем произведения, оканчивающиеся на "ов"
        result = task_3(self.many_to_many, 'ов')

        # Должен найтись только "Марш Петров"
        self.assertEqual(len(result), 1)
        self.assertEqual(result[0][0], 'Марш Петров')
        self.assertEqual(result[0][1], 'Оркестр В')


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

Скриншот работы программы

```
philippschepnov@MacBook-Pro-Filipp-2 rk2 % python3 main.py
Задание Б1
Концерт для скрипки - Камерный оркестр
Рапсодия в стиле блюз - Джазовый оркестр
Симфониетта - Камерный оркестр
Симфония №5 - Симфонический оркестр
Симфония №9 - Симфонический оркестр
Увертюра 1812 - Симфонический оркестр

Задание Б2
Симфонический оркестр: 3 произведений
Камерный оркестр: 2 произведений
Джазовый оркестр: 1 произведений
Народный оркестр: 0 произведений

Задание Б3
philippschepnov@MacBook-Pro-Filipp-2 rk2 % python3 test_rk2.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```