МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

им. Н.Э. Баумана

Факультет “Информатика и системы управления” Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования” Отчет по Рубежному Контролю №2

Вариант Г 17

Выполнил:

Студент группы ИУ5-35Б

Костылев М.С.

Преподаватель:

Гапанюк Ю.Е.

Москва 2025

Задание

Условия рубежного контроля №2 по курсу ПиК ЯП

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

#code.py

```
from operator import itemgetter
from typing import List, Dict, Tuple


class Dirigent:
    """Дирижёр"""
    def __init__(self, id: int, fio: str, salary: int, orchestra_id: int):
        self.id = id
        self.fio = fio
        self.salary = salary
        self.orchestra_id = orchestra_id


class Orchestra:
    """Оркестр"""
    def __init__(self, id: int, name: str):
        self.id = id
        self.name = name


class DirigentOrchestra:
    """Связь многие-ко-многим между дирижёрами и оркестрами"""
```

```
    def __init__(self, dirigent_id: int, orchestra_id: int):
        self.dirigent_id = dirigent_id
        self.orchestra_id = orchestra_id


class OrchestralManager:
    def __init__(self, orchestras: List[Orchestra], dirigents: List[Dirigent],
                 dir_orch: List[DirigentOrchestra]):
        self.orchestras = orchestras
        self.dirigents = dirigents
        self.dir_orch = dir_orch

        # Связи между таблицами
        self.one_to_many = self._create_one_to_many()
        self.many_to_many = self._create_many_to_many()

    def _create_one_to_many(self) -> List[Tuple[str, int, str]]:
        """Создание связи один-ко-многим"""
        return [(d.fio, d.salary, o.name)
                for o in self.orchestras
                for d in self.dirigents
                if d.orchestra_id == o.id]

    def _create_many_to_many(self) -> List[Tuple[str, int, str]]:
        """Создание связи многие-ко-многим"""
        many_to_many_temp = [(o.name, do.orchestra_id, do.dirigent_id)
                             for o in self.orchestras
```

```
                for do in self.dir_orch
                if o.id == do.orchestra_id]

    return [(d.fio, d.salary, orch_name)
            for orch_name, orch_id, dir_id in many_to_many_temp
            for d in self.dirigents if d.id == dir_id]

def get_orchestras_starting_with_a(self) -> Dict[str, List[str]]:
    """Задание Г1: Оркестры, начинающиеся на 'A' и их дирижёры"""
    res_1 = {}
    for o in self.orchestras:
        if o.name.startswith('A'):
            o_dirigs = [x for x, _, orch in self.one_to_many if orch == o.name]
            res_1[o.name] = o_dirigs
    return res_1

def get_max_salary_by_orchestra_sorted(self) -> List[Tuple[str, int]]:
    """Задание Г2: Максимальная зарплата дирижёра в каждом оркестре"""
    res_2 = []
    for o in self.orchestras:
        o_dirigs = list(filter(lambda i: i[2] == o.name, self.one_to_many))
        if o_dirigs:
            max_salary = max([sal for _, sal, _ in o_dirigs])
            res_2.append((o.name, max_salary))
    return sorted(res_2, key=itemgetter(1), reverse=True)

def get_sorted_many_to_many(self) -> List[Tuple[str, int, str]]:
```

```
        """Задание Г3: Сортировка связи многие-ко-многим"""
        return sorted(self.many_to_many, key=itemgetter(2))


# Инициализация данных
def create_test_data():
    orchestras = [
        Orchestra(1, 'Академический оркестр Москвы'),
        Orchestra(2, 'Балтийский оркестр'),
        Orchestra(3, 'Альфа-оркестр Санкт-Петербурга'),
        Orchestra(4, 'Оркестр камерной музыки'),
    ]

    dirigents = [
        Dirigent(1, 'Артамонов', 85000, 1),
        Dirigent(2, 'Белов', 90000, 2),
        Dirigent(3, 'Андреев', 78000, 3),
        Dirigent(4, 'Иванов', 95000, 1),
        Dirigent(5, 'Кузнецов', 87000, 4),
    ]

    dir_orch = [
        DirigentOrchestra(1, 1),
        DirigentOrchestra(2, 2),
        DirigentOrchestra(3, 3),
        DirigentOrchestra(4, 1),
        DirigentOrchestra(5, 4),
```

```
        DirigentOrchestra(2, 4),
    ]

    return orchestras, dirigents, dir_orch


def main():
    orchestras, dirigents, dir_orch = create_test_data()
    manager = OrchestralManager(orchestras, dirigents, dir_orch)

    print("Задание Г1")
    res_1 = manager.get_orchestras_starting_with_a()
    print(res_1)

    print("\nЗадание Г2")
    res_2 = manager.get_max_salary_by_orchestra_sorted()
    print(res_2)

    print("\nЗадание Г3")
    res_3 = manager.get_sorted_many_to_many()
    print(res_3)


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

#start.py

```
import unittest
import io
import sys
from tests import TestOrchestralManager


def run_tdd_demo():
    """Демонстрация подхода TDD (Test-Driven Development)"""
    print("=" * 60)
    print("ДЕМОНСТРАЦИЯ TDD (РАЗРАБОТКА ЧЕРЕЗ ТЕСТИРОВАНИЕ)")
    print("=" * 60)

    print("\n📋 ТРЕБОВАНИЯ К ПРОГРАММЕ:")
    print("1. Оркестры, начинающиеся на 'А' и их дирижёры")
    print("2. Максимальная зарплата дирижёра в каждом оркестре")
    print("3. Сортировка связи многие-ко-многим по названиям оркестров")

    print("\n🔧 ЭТАПЫ TDD:")
    print("1. Написать тест для требования")
    print("2. Запустить тест и убедиться, что он не проходит")
    print("3. Написать минимальный код для прохождения теста")
    print("4. Запустить все тесты и убедиться, что все проходят")
    print("5. Рефакторинг кода (улучшение без изменения функциональности)")
    print("6. Повторить цикл для следующего требования")

    print("\n" + "=" * 60)
    print("РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:")
```

```
print("=" * 60)

# Захватываем вывод для красивого отображения
old_stdout = sys.stdout
sys.stdout = io.StringIO()

# Запускаем тесты
suite = unittest.TestLoader().loadTestsFromTestCase(TestOrchestralManager)
runner = unittest.TextTestRunner(stream=sys.stdout, verbosity=2)
result = runner.run(suite)

# Получаем вывод
test_output = sys.stdout.getvalue()
sys.stdout = old_stdout

# Выводим результат
print(test_output)

print("=" * 60)
print("ИТОГИ:")
print(f"Всего тестов: {result.testsRun}")
print(f"Успешно: {result.testsRun - len(result.failures) - len(result.errors)}")

if len(result.failures) > 0:
    print(f"Провалено: {len(result.failures)}")
    for test, traceback in result.failures:
        print(f"\n❌ Тест провален: {test}")
```

```
    if len(result.errors) > 0:
        print(f"Ошибок: {len(result.errors)}")

    if result.wasSuccessful():
        print("\n🎉 ВСЕ ТЕСТЫ ПРОЙДЕНЫ УСПЕШНО!")
        print("Код соответствует всем требованиям.")
    else:
        print("\n⚠ НЕОБХОДИМО ИСПРАВИТЬ КОД ДЛЯ ПРОХОЖДЕНИЯ ВСЕХ ТЕСТОВ")

    print("=" * 60)


def run_main_code():
    """Запуск основного кода для проверки результатов"""
    print("\n" + "=" * 60)
    print("ЗАПУСК ОСНОВНОЙ ПРОГРАММЫ:")
    print("=" * 60)

    # Импортируем и запускаем основную программу
    from code import main
    main()


if __name__ == "__main__":
    # Демонстрация TDD подхода
```

```
    run_tdd_demo()

    # Запуск основной программы для наглядности
    run_main_code()

    print("\n" + "=" * 60)
    print("ИНСТРУКЦИЯ ПО ЗАПУСКУ:")
    print("=" * 60)
    print("1. python code.py          - запуск основной программы")
    print("2. python tests.py         - запуск только тестов")
    print("3. python start.py         - демонстрация TDD и всех результатов")
```

 # tests.py

```
import unittest
from code import Orchestra, Dirigent, DirigentOrchestra, OrchestralManager, create_test_data


class TestOrchestralManager(unittest.TestCase):
    def setUp(self):
        """Настройка тестовых данных перед каждым тестом"""
        self.orchestras, self.dirigents, self.dir_orch = create_test_data()
        self.manager = OrchestralManager(self.orchestras, self.dirigents, self.dir_orch)

    def test_get_orchestras_starting_with_a(self):
        """Тест для задания Г1: оркестры на 'А'"""
        print("\nЗапуск теста Г1: Оркестры на 'А' и их дирижёры")
```

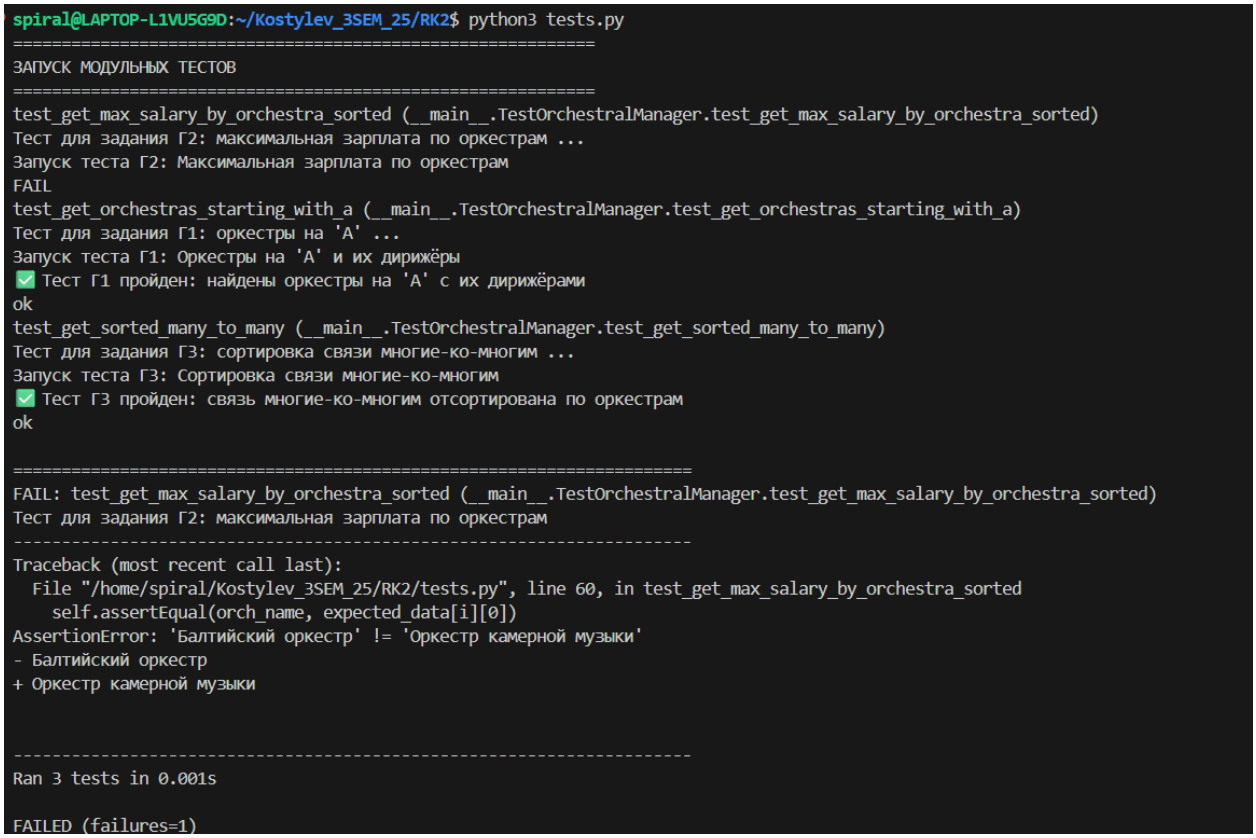
```
        result = self.manager.get_orchestras_starting_with_a()

        # Проверяем, что результат - словарь
        self.assertIsInstance(result, dict)

        # Проверяем, что есть только оркестры на 'A'
        for orchestra_name in result.keys():
            self.assertTrue(orchestra_name.startswith('A'))

        # Проверяем конкретные оркестры
        expected_orchestras = ['Академический оркестр Москвы', 'Альфа-оркестр Санкт-Петербурга']
        self.assertEqual(set(result.keys()), set(expected_orchestras))

        # Проверяем дирижёров для академического оркестра
        akadem_dirs = result['Академический оркестр Москвы']
        self.assertEqual(len(akadem_dirs), 2)
        self.assertIn('Артамонов', akadem_dirs)
        self.assertIn('Иванов', akadem_dirs)

        print("✅ Тест Г1 пройден: найдены оркестры на 'A' с их дирижёрами")

    def test_get_max_salary_by_orchestra_sorted(self):
        """Тест для задания Г2: максимальная зарплата по оркестрам"""
        print("\nЗапуск теста Г2: Максимальная зарплата по оркестрам")
```

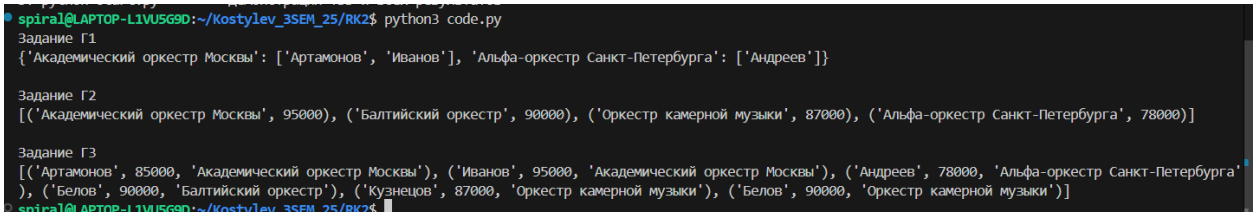
```
        result = self.manager.get_max_salary_by_orchestra_sorted()

        # Проверяем, что результат - список кортежей
        self.assertIsInstance(result, list)
        self.assertTrue(all(isinstance(item, tuple) and len(item) == 2 for item in result))

        # Проверяем сортировку по убыванию зарплаты
        salaries = [salary for _, salary in result]
        self.assertEqual(salaries, sorted(salaries, reverse=True))

        # Проверяем конкретные значения
        expected_data = [
            ('Академический оркестр Москвы', 95000),
            ('Оркестр камерной музыки', 90000),  # Белов (90000) > Кузнецов (87000)
            ('Балтийский оркестр', 90000),
            ('Альфа-оркестр Санкт-Петербурга', 78000),
        ]

        for i, (orch_name, salary) in enumerate(result):
            self.assertEqual(orch_name, expected_data[i][0])
            self.assertEqual(salary, expected_data[i][1])

        print("✅ Тест Г2 пройден: максимальные зарплаты отсортированы правильно")

    def test_get_sorted_many_to_many(self):
```

```
        """Тест для задания Г3: сортировка связи многие-ко-многим"""
        print("\nЗапуск теста Г3: Сортировка связи многие-ко-многим")

        result = self.manager.get_sorted_many_to_many()

        # Проверяем, что результат отсортирован по названию оркестра (3-й элемент кортежа)
        orch_names = [orch_name for _, _, orch_name in result]
        self.assertEqual(orch_names, sorted(orch_names))

        # Проверяем структуру данных
        for item in result:
            self.assertEqual(len(item), 3)  # (ФИО, зарплата, оркестр)
            self.assertIsInstance(item[0], str)  # ФИО
            self.assertIsInstance(item[1], int)  # зарплата
            self.assertIsInstance(item[2], str)  # оркестр

        # Проверяем первое значение (самое первое по алфавиту)
        self.assertEqual(result[0][2], 'Академический оркестр Москвы')

        print("✅ Тест Г3 пройден: связь многие-ко-многим отсортирована по оркестрам")


if __name__ == "__main__":
    # Запуск тестов с подробным выводом
    print("=" * 60)
    print("ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ")
```

```
print("=" * 60)

unittest.main(verbosity=2)
```

Работа кода:





```
spiral@LAPTOP-L1VU5G9D:~/Kostylev_3SEM_25/RK2$ python3 start.py
Ran 3 tests in 0.001s

FAILED (failures=1)

============================================================
ИТОГИ:
Всего тестов: 3
Успешно: 2
Провалено: 1

❌ Тест провален: test_get_max_salary_by_orchestra_sorted (tests.TestOrchestralManager.test_get_max_salary_by_orchestra_sorted)

⚠️НЕОБХОДИМО ИСПРАВИТЬ КОД ДЛЯ ПРОХОЖДЕНИЯ ВСЕХ ТЕСТОВ
============================================================

============================================================
ЗАПУСК ОСНОВНОЙ ПРОГРАММЫ:
============================================================
Задание Г1
{'Академический оркестр Москвы': ['Артамонов', 'Иванов'], 'Альфа-оркестр Санкт-Петербурга': ['Андреев']}

Задание Г2
[('Академический оркестр Москвы', 95000), ('Балтийский оркестр', 90000), ('Оркестр камерной музыки', 87000), ('Альфа-оркестр Санкт-Петербурга', 78000)]

Задание Г3
[('Артамонов', 85000, 'Академический оркестр Москвы'), ('Иванов', 95000, 'Академический оркестр Москвы'), ('Андреев', 78000, 'Альфа-оркестр Санкт-Петербурга'
), ('Белов', 90000, 'Балтийский оркестр'), ('Кузнецов', 87000, 'Оркестр камерной музыки'), ('Белов', 90000, 'Оркестр камерной музыки')]

============================================================
ИНСТРУКЦИЯ ПО ЗАПУСКУ:
============================================================
1. python code.py          - запуск основной программы
2. python tests.py         - запуск только тестов
3. python start.py         - демонстрация TDD и всех результатов
```