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 «ИНЖЕНЕРНЫЙ БИЗНЕС И МЕНЕДЖМЕНТ»

КАФЕДРА «ПРОМЫШЛЕННАЯ ЛОГИСТИКА» (ИБМ-3)

Отчёт

По рубежному контролю №2

По дисциплине «Парадигмы и конструкции языков программирования»

Вариант 18

Студент ИБМ3-34Б:

Нургалиева А.А.

Преподаватель:

Гапанюк Ю.Е.

2024 г.

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).

1) Проведём рефакторинг теста программы:

```
# Рефакторинг программы для модульного тестирования

class MusicalPiece:
    def __init__(self, id, title, composer, duration, genre):
        self.id = id
        self.title = title
        self.composer = composer
        self.duration = duration
        self.genre = genre
        self.orchestral_ids = []  # Список ID оркестров, исполняющих
произведение


class Orchestra:
    def __init__(self, id, name, location):
        self.id = id
        self.name = name
        self.location = location
        self.piece_ids = []  # Список ID музыкальных произведений,
исполняемых оркестром


class Performance:
    def __init__(self, piece_id, orchestra_id):
        self.piece_id = piece_id
        self.orchestra_id = orchestra_id


def establish_relationships(pieces, orchestras, performances):
    """Создает связи многие-ко-многим между объектами."""
    for p in pieces:
        for perf in performances:
            if perf.piece_id == p.id:
                p.orchestral_ids.append(perf.orchestra_id)

    for o in orchestras:
        for perf in performances:
            if perf.orchestra_id == o.id:
                o.piece_ids.append(perf.piece_id)


def get_pieces_by_orchestras(pieces, orchestras):
    """Возвращает список произведений и оркестров, отсортированный по
названию оркестра."""
    from operator import itemgetter

    one_to_many = [(p.title, p.duration, o.name)
```

```
                  for o in orchestras
                  for p in pieces
                  if p.id in o.piece_ids]

    return sorted(one_to_many, key=itemgetter(2))


def get_orchestras_total_duration(pieces, orchestras):
    """Возвращает список оркестров с суммарной длительностью их
произведений."""
    res = []
    for o in orchestras:
        o_pieces = [(p.title, p.duration, o.name)
                    for p in pieces
                    if p.id in o.piece_ids]
        if o_pieces:
            total_duration = sum(p[1] for p in o_pieces)
            res.append((o.name, total_duration))

    return sorted(res, key=lambda x: x[1])


def get_orchestras_with_keyword(pieces, orchestras, keyword):
    """Возвращает список оркестров, содержащих keyword в названии, и их
произведений."""
    return [(o.name, [p.title for p in pieces if p.id in o.piece_ids])
            for o in orchestras if keyword.lower() in o.name.lower()]
```

2) Создадим модульные тесты с применением TDD-фреймворка:

```
# Данные для тестов
pieces = [
    MusicalPiece(1, 'Симфония №5', 'Бетховен', 45, 'Классика'),
    MusicalPiece(2, 'Кармен', 'Бизе', 60, 'Опера'),
    MusicalPiece(3, 'Болеро', 'Равель', 15, 'Классика'),
    MusicalPiece(4, 'Времена года', 'Вивальди', 30, 'Классика')
]

orchestras = [
    Orchestra(1, 'Филармонический оркестр', 'Москва'),
    Orchestra(2, 'Симфонический оркестр', 'Санкт-Петербург'),
    Orchestra(3, 'Камерный оркестр', 'Мюнхен')
]

performances = [
    Performance(1, 1),
    Performance(1, 2),
    Performance(2, 1),
    Performance(3, 3),
    Performance(4, 2)
]

establish_relationships(pieces, orchestras, performances)


# Модульные тесты
import unittest


class TestMusicProgram(unittest.TestCase):

    def setUp(self):
```

```
        self.pieces = pieces
        self.orchestras = orchestras
        self.performances = performances
        establish_relationships(self.pieces, self.orchestras,
self.performances)

    def test_get_pieces_by_orchestras(self):
        result = get_pieces_by_orchestras(self.pieces, self.orchestras)
        expected = [
            ('Болеро', 15, 'Камерный оркестр'),
            ('Симфония №5', 45, 'Симфонический оркестр'),
            ('Времена года', 30, 'Симфонический оркестр'),
            ('Симфония №5', 45, 'Филармонический оркестр'),
            ('Кармен', 60, 'Филармонический оркестр')
        ]
        self.assertEqual(result, expected)

    def test_get_orchestras_total_duration(self):
        result = get_orchestras_total_duration(self.pieces, self.orchestras)
        expected = [
            ('Камерный оркестр', 15),
            ('Симфонический оркестр', 75),
            ('Филармонический оркестр', 105)
        ]
        self.assertEqual(result, expected)

    def test_get_orchestras_with_keyword(self):
        result = get_orchestras_with_keyword(self.pieces, self.orchestras,
'оркестр')
        expected = [
            ('Филармонический оркестр', ['Симфония №5', 'Кармен']),
            ('Симфонический оркестр', ['Симфония №5', 'Времена года']),
            ('Камерный оркестр', ['Болеро'])
        ]
        self.assertEqual(result, expected)


if __name__ == '__main__':
    unittest.main()
Refactoring the program for modular testing
```

Тест проведён: