**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 запросов: E
Вариант предметной области: 14

Выполнил:
студент группы РТ5-31Б
Романов Ю.И.
Подпись и дата:

Проверил:
к.т.н., доц. каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва, 2025 г.

## Описание задания

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).

## Текст программы

**main.py:**

```
from operator import itemgetter

# CD-диск
class CD:
    def __init__(self, id, title, capacity, lib_id):
        self.id = id
        self.title = title
        self.capacity = capacity    # объем в МБ
        self.lib_id = lib_id

# Библиотека CD-дисков
class CDLibrary:
    def __init__(self, id, name):
        self.id = id
        self.name = name

# CD-диски в библиотеках для реализации связи многие-ко-многим
class CD_CDLibrary:
    def __init__(self, lib_id, cd_id):
        self.lib_id = lib_id
        self.cd_id = cd_id


class CDLibrarySystem:
    def __init__(self, libraries, cds, cds_libraries):
        self.libraries = libraries
        self.cds = cds
        self.cds_libraries = cds_libraries
        self.one_to_many = None
```

```
        self.many_to_many = None
        self._prepare_data()

    def _prepare_data(self):
        # Соединение данных один-ко-многим
        self.one_to_many = [(cd.title, cd.capacity, lib.name)
                            for lib in self.libraries
                            for cd in self.cds
                            if cd.lib_id == lib.id]

        # Соединение данных многие-ко-многим
        many_to_many_temp = [(lib.name, cl.lib_id, cl.cd_id)
                             for lib in self.libraries
                             for cl in self.cds_libraries
                             if lib.id == cl.lib_id]

        self.many_to_many = [(cd.title, cd.capacity, lib_name)
                             for lib_name, _, cd_id in many_to_many_temp
                             for cd in self.cds if cd.id == cd_id]

    def get_libraries_with_name_containing(self, search_term):
        """Задание E1: Библиотеки, содержащие search_term в названии, с их
CD-дисками"""
        result = {}
        for lib in self.libraries:
            if search_term in lib.name:
                lib_cds = list(filter(lambda i: i[2] == lib.name, self.one_to_many))
                lib_cds_titles = [x for x, _, _ in lib_cds]
                result[lib.name] = lib_cds_titles
        return result

    def get_avg_capacity_by_library(self):
        """Задание E2: Библиотеки со средней емкостью CD-дисков,
отсортированные по средней емкости"""
        result_unsorted = []
        for lib in self.libraries:
            lib_cds = list(filter(lambda i: i[2] == lib.name, self.one_to_many))
            if len(lib_cds) > 0:
                lib_capacities = [capacity for _, capacity, _ in lib_cds]
                lib_avg_capacity = round(sum(lib_capacities) / len(lib_capacities), 2)
                result_unsorted.append((lib.name, lib_avg_capacity))

        # Сортировка по средней емкости
        return sorted(result_unsorted, key=itemgetter(1))
```

```
    def get_cds_starting_with_letter(self, letter):
        """Задание E3: CD-диски, название которых начинается с заданной буквы"""
        filtered_cds = list(filter(lambda i: i[0].startswith(letter), self.many_to_many))
        # Сортировка по названию CD-диска
        return sorted(filtered_cds, key=itemgetter(0))

def create_sample_data():
    """Создание тестовых данных"""
    # Библиотеки CD-дисков
    libraries = [
        CDLibrary(1, 'библиотека мультимедиа'),
        CDLibrary(2, 'архив программного обеспечения'),
        CDLibrary(3, 'библиотека фильмов'),
        CDLibrary(4, 'основной архив'),
        CDLibrary(5, 'коллекция литературы'),
        CDLibrary(6, 'библиотека игр'),
    ]

    # CD-диски
    cds = [
        CD(11, 'Альбом Rock', 700, 1),
        CD(22, 'Мастер и Маргарита', 650, 5),
        CD(33, 'Фильм "Гарри Поттер и Орден Феникса"', 900, 3),
        CD(44, 'Minecraft', 700, 6),
        CD(55, 'Antivirus Pro', 500, 2),
        CD(66, 'Microsoft Office', 650, 4),
    ]

    # Связи многие-ко-многим
    cds_libraries = [
        CD_CDLibrary(1, 11),
        CD_CDLibrary(1, 22),
        CD_CDLibrary(1, 33),
        CD_CDLibrary(2, 44),
        CD_CDLibrary(2, 55),
        CD_CDLibrary(2, 66),
        CD_CDLibrary(3, 33),
        CD_CDLibrary(4, 55),
        CD_CDLibrary(4, 66),
        CD_CDLibrary(5, 22),
        CD_CDLibrary(6, 44)
    ]

    return libraries, cds, cds_libraries
```

```
def main():
    libraries, cds, cds_libraries = create_sample_data()
    system = CDLibrarySystem(libraries, cds, cds_libraries)

    print('Задание E1')
    res_e1 = system.get_libraries_with_name_containing('библиотека')
    print(res_e1)

    print('\nЗадание E2')
    res_e2 = system.get_avg_capacity_by_library()
    print(res_e2)

    print('\nЗадание E3')
    res_e3 = system.get_cds_starting_with_letter('Ф')
    print(res_e3)

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

**test_cd_library.py:**

```
import unittest
from main import CDLibrarySystem, create_sample_data

class TestCDLibrarySystem(unittest.TestCase):
    def setUp(self):
        """Настройка тестовых данных перед каждым тестом"""
        libraries, cds, cds_libraries = create_sample_data()
        self.system = CDLibrarySystem(libraries, cds, cds_libraries)

    def test_get_libraries_with_name_containing(self):
        """Тест для задания E1: поиск библиотек по названию"""
        # Ищем библиотеки, содержащие "библиотека" в названии
        result = self.system.get_libraries_with_name_containing('библиотека')

        # Проверяем, что найдены правильные библиотеки
        expected_libraries = {'библиотека мультимедиа', 'библиотека фильмов',
'библиотека игр'}
        self.assertEqual(set(result.keys()), expected_libraries)

        # Проверяем количество CD-дисков в библиотеке мультимедиа
        self.assertEqual(len(result['библиотека мультимедиа']), 1)

        # Проверяем конкретные CD-диски в библиотеке игр
```

```
        self.assertIn('Minecraft', result['библиотека игр'])

    def test_get_avg_capacity_by_library(self):
        """Тест для задания E2: средняя емкость CD-дисков по библиотекам"""
        result = self.system.get_avg_capacity_by_library()

        # Проверяем, что результат отсортирован по средней емкости
        capacities = [avg for _, avg in result]
        self.assertEqual(capacities, sorted(capacities))

        # Проверяем правильность расчета для библиотеки мультимедиа
        for lib_name, avg_capacity in result:
            if lib_name == 'библиотека мультимедиа':
                self.assertEqual(avg_capacity, 700.0)
                break

        # Проверяем, что все библиотеки присутствуют в результате
        self.assertEqual(len(result), 6)

    def test_get_cds_starting_with_letter(self):
        """Тест для задания E3: поиск CD-дисков по начальной букве"""
        # Ищем CD-диски, начинающиеся с 'Ф' (кириллица)
        result = self.system.get_cds_starting_with_letter('Ф')

        # Проверяем, что найден правильный CD-диск
        self.assertEqual(len(result), 2)
        self.assertEqual(result[0][0], 'Фильм "Гарри Поттер и Орден Феникса"')

        # Проверяем, что результат отсортирован по названию
        titles = [title for title, _, _ in result]
        self.assertEqual(titles, sorted(titles))

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

**run_tests.py:**

```
import unittest
from test_cd_library import TestCDLibrarySystem

if __name__ == '__main__':
    suite = unittest.TestLoader().loadTestsFromTestCase(TestCDLibrarySystem)

    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)
```

```
print(f"\n{'='*70}")
print(f"Всего тестов: {result.testsRun}")
print(f"Провалено: {len(result.failures)}")
print(f"Ошибок: {len(result.errors)}")
if result.wasSuccessful():
    print("Все тесты пройдены успешно!")
```

## Экранные формы с примерами выполнения программы