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
Ковригина Д. М.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

Данный отчёт описывает выполнение рубежного контроля №2 по курсу "Программирование и компьютерные практикумы". Работа состояла из двух основных частей:

1. Рефакторинг исходного кода программы рубежного контроля №1 для обеспечения пригодности к модульному тестированию.
2. Создание модульных тестов с применением TDD-подхода (3 теста).

2. Исходный код

2.1 Анализ исходного кода (RK1.py)

Исходная программа представляла собой модуль для обработки данных о школьниках и классах с тремя типами запросов:

Структура данных:

- Класс Student (школьник) - id, ФИО, оценка, id класса
- Класс SchoolClass (школьный класс) - id, название
- Класс StudentClass (связь школьник-класс) - id класса, id школьника

Функциональность:

1. **Запрос Б1**: Список связанных школьников и классов (1:M), отсортированный по школьникам
2. **Запрос Б2**: Список классов с количеством школьников, отсортированный по количеству
3. **Запрос Б3**: Список школьников с фамилиями на "ов" и их классов (M:M)

3. Рефакторинг

3.1 Принципы рефакторинга

Для обеспечения тестируемости код был рефакторирован с применением следующих принципов:

1. **Инкапсуляция данных и логики**
2. **Разделение ответственности (Single Responsibility Principle)**
3. **Выделение тестируемых единиц**
4. **Документирование интерфейсов**

3.2 Структура рефакторингованного кода

```
class Student:

    def __init__(self, id, fio, grade, class_id):
        self.id = id
        self.fio = fio
        self.grade = grade
        self.class_id = class_id

    def __repr__(self):
        return f"Student(id={self.id}, fio='{self.fio}', grade={self.grade},
class_id={self.class_id})"


class SchoolClass:

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

    def __repr__(self):
        return f"SchoolClass(id={self.id}, name='{self.name}')"


class StudentClass:

    def __init__(self, class_id, student_id):
        self.class_id = class_id
        self.student_id = student_id

    def __repr__(self):
        return f"StudentClass(class_id={self.class_id},
student_id={self.student_id})"


class SchoolDataProcessor:

    def __init__(self, classes, students, students_classes):
        self.classes = classes
        self.students = students
        self.students_classes = students_classes
```

```python
    def get_one_to_many_data(self):
        return [
            [stud.fio, stud.grade, cl.name]
            for stud in self.students
            for cl in self.classes
            if stud.class_id == cl.id
        ]

    def get_classes_with_student_count(self):
        one_to_many = self.get_one_to_many_data()
        result = []

        for cl in self.classes:
            studs_in_class = list(filter(lambda x: x[2] == cl.name, one_to_many))
            if studs_in_class:
                result.append((cl.name, len(studs_in_class)))

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

    def get_many_to_many_data(self):
        many_to_many_first = [
            [cl.name, sc.class_id, sc.student_id]
            for cl in self.classes
            for sc in self.students_classes
            if cl.id == sc.class_id
        ]

        return [
            [stud.fio, class_name]
            for class_name, class_id, stud_id in many_to_many_first
            for stud in self.students
            if stud.id == stud_id
        ]

    def get_students_with_ov_ending(self):
        many_to_many = self.get_many_to_many_data()
        result = []

        for fio, class_name in many_to_many:
            if fio.endswith("ов"):
                result.append([fio, class_name])

        return sorted(result, key=lambda x: x[0])

    def get_students_sorted_by_name(self):
        one_to_many = self.get_one_to_many_data()
        return sorted(one_to_many, key=lambda x: x[0])


def main():
    classes = [
```

```
        SchoolClass(1, "7А"),
        SchoolClass(2, "7Б"),
        SchoolClass(3, "8В"),
        SchoolClass(4, "8Г"),
    ]

    students = [
        Student(1, "Иванов", 4.5, 1),
        Student(2, "Петров", 3.8, 2),
        Student(3, "Сидоров", 4.2, 3),
        Student(4, "Кузнец", 4.8, 3),
        Student(5, "Никитин", 3.9, 3),
        Student(6, "Беляев", 4.1, 4),
    ]

    students_classes = [
        StudentClass(1, 1),
        StudentClass(3, 2),
        StudentClass(3, 3),
        StudentClass(3, 4),
        StudentClass(2, 5),
        StudentClass(4, 6),
        StudentClass(4, 2),
        StudentClass(2, 1),
    ]

    processor = SchoolDataProcessor(classes, students, students_classes)

    print("--- Запрос Б1 ---")
    print("Список всех связанных школьников и классов (1:М), отсортированный по
школьникам:")
    arr1 = processor.get_students_sorted_by_name()
    for item in arr1:
        print(f" Школьник: {item[0]}, Оценка: {item[1]}, Класс: {item[2]}")

    print("\n--- Запрос Б2 ---")
    print("Список классов с количеством школьников в каждом (1:М),
отсортированный по количеству (по возрастанию):")
    arr2 = processor.get_classes_with_student_count()
    for item in arr2:
        print(f" Класс: {item[0]}, Количество школьников: {item[1]}")

    print("\n--- Запрос Б3 ---")
    print("Список всех школьников, у которых фамилия заканчивается на 'ов', и
названия их классов (М:М):")
    arr3 = processor.get_students_with_ov_ending()
    for item in arr3:
        print(f" Школьник: {item[0]}, Класс: {item[1]}")


if __name__ == "__main__":
```

```
    main()
```

4. Модульное тестирование

4.1 Подход TDD (Test-Driven Development)

Для создания тестов применялась методология TDD, которая предполагает:

1. Написание тестов до реализации функциональности
2. Малые инкрементальные изменения
3. Постоянный рефакторинг

4.2 Структура тестов

```
import unittest
from refactored_RK1 import Student, SchoolClass, StudentClass,
SchoolDataProcessor


class TestSchoolDataProcessor(unittest.TestCase):

    def setUp(self):
        self.classes = [
            SchoolClass(1, "7А"),
            SchoolClass(2, "7Б"),
            SchoolClass(3, "8В"),
            SchoolClass(4, "8Г"),
        ]

        self.students = [
            Student(1, "Иванов", 4.5, 1),
            Student(2, "Петров", 3.8, 2),
            Student(3, "Сидоров", 4.2, 3),
            Student(4, "Кузнец", 4.8, 3),
            Student(5, "Никитин", 3.9, 3),
            Student(6, "Беляев", 4.1, 4),
        ]

        self.students_classes = [
            StudentClass(1, 1),
            StudentClass(3, 2),
            StudentClass(3, 3),
            StudentClass(3, 4),
            StudentClass(2, 5),
            StudentClass(4, 6),
```

```
            StudentClass(4, 2),
            StudentClass(2, 1),
        ]

        self.processor = SchoolDataProcessor(
            self.classes,
            self.students,
            self.students_classes
        )

    def test_get_one_to_many_data(self):
        result = self.processor.get_one_to_many_data()

        self.assertEqual(len(result), 6, "Неверное количество записей 1:М")

        for item in result:
            self.assertEqual(len(item), 3, "Неверная структура записи")
            self.assertIsInstance(item[0], str, "ФИО должно быть строкой")
            self.assertIsInstance(item[1], float, "Оценка должна быть числом")
            self.assertIsInstance(item[2], str, "Название класса должно быть
строкой")

        student_names = [item[0] for item in result]
        self.assertIn("Иванов", student_names, "Иванов должен быть в списке")
        self.assertIn("Петров", student_names, "Петров должен быть в списке")

    def test_get_classes_with_student_count(self):
        """Тест 2: Проверка подсчета школьников по классам."""
        result = self.processor.get_classes_with_student_count()

        counts = [count for _, count in result]
        self.assertEqual(counts, sorted(counts), "Список должен быть отсортирован
по количеству")

        for class_name, count in result:
            self.assertIsInstance(class_name, str, "Название класса должно быть
строкой")
            self.assertIsInstance(count, int, "Количество должно быть целым
числом")
            self.assertGreaterEqual(count, 0, "Количество не может быть
отрицательным")

        class_dict = dict(result)
        self.assertEqual(class_dict.get("8В"), 3, "В классе 8В должно быть 3
школьника")
        self.assertEqual(class_dict.get("7А"), 1, "В классе 7А должен быть 1
школьник")

    def test_get_students_with_ov_ending(self):
        """Тест 3: Проверка поиска школьников с фамилиями на 'ов'."""
        result = self.processor.get_students_with_ov_ending()
```

```
        surnames = [item[0] for item in result]
        self.assertEqual(surnames, sorted(surnames), "Список должен быть
отсортирован по фамилии")

        for surname, _ in result:
            self.assertTrue(surname.endswith("ов"), f"Фамилия '{surname}' должна
заканчиваться на 'ов'")

        for surname, class_name in result:
            self.assertIsInstance(surname, str, "Фамилия должна быть строкой")
            self.assertIsInstance(class_name, str, "Название класса должно быть
строкой")

        found_surnames = [item[0] for item in result]
        self.assertIn("Иванов", found_surnames, "Иванов должен быть найден")
        self.assertIn("Петров", found_surnames, "Петров должен быть найден")
        self.assertIn("Сидоров", found_surnames, "Сидоров должен быть найден")
        self.assertNotIn("Кузнец", found_surnames, "Кузнец не должен быть
найден")

    def test_edge_cases(self):
        empty_processor = SchoolDataProcessor([], [], [])
        self.assertEqual(empty_processor.get_one_to_many_data(), [])
        self.assertEqual(empty_processor.get_classes_with_student_count(), [])
        self.assertEqual(empty_processor.get_students_with_ov_ending(), [])

        self.assertEqual(empty_processor.get_students_sorted_by_name(), [])

    def test_data_consistency(self):
        one_to_many = self.processor.get_one_to_many_data()
        class_names = {cl.name for cl in self.classes}

        for _, _, class_name in one_to_many:
            self.assertIn(class_name, class_names, f"Класс '{class_name}' не
существует")

        many_to_many = self.processor.get_many_to_many_data()
        student_ids = {stud.id for stud in self.students}

        many_to_many_first = [
            [cl.name, sc.class_id, sc.student_id]
            for cl in self.classes
            for sc in self.students_classes
            if cl.id == sc.class_id
        ]

        for _, class_id, student_id in many_to_many_first:
            self.assertIn(student_id, student_ids, f"Студент с id={student_id} не
существует")
```

```
            self.assertIn(class_id, [cl.id for cl in self.classes], f"Класс с
id={class_id} не существует")


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

Тесты успешно проходят:

```
text
test_data_consistency ... ok
test_edge_cases ... ok
test_get_classes_with_student_count ... ok
test_get_one_to_many_data ... ok
test_get_students_with_ov_ending ... ok
----------------------------------------------------------------------
Ran 5 tests in 0.001s
OK
```