РК2

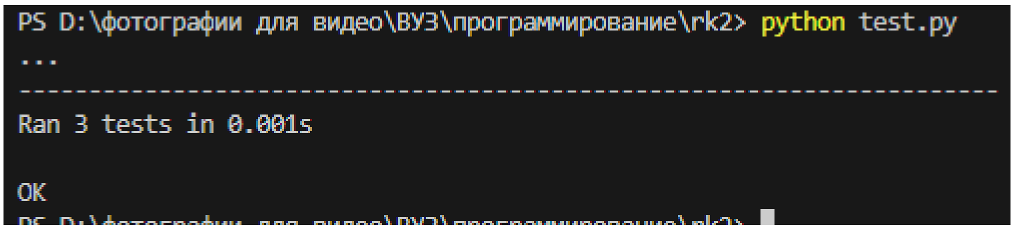
Тенишев Александр, ИУ5-35Б

Описание задания

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).

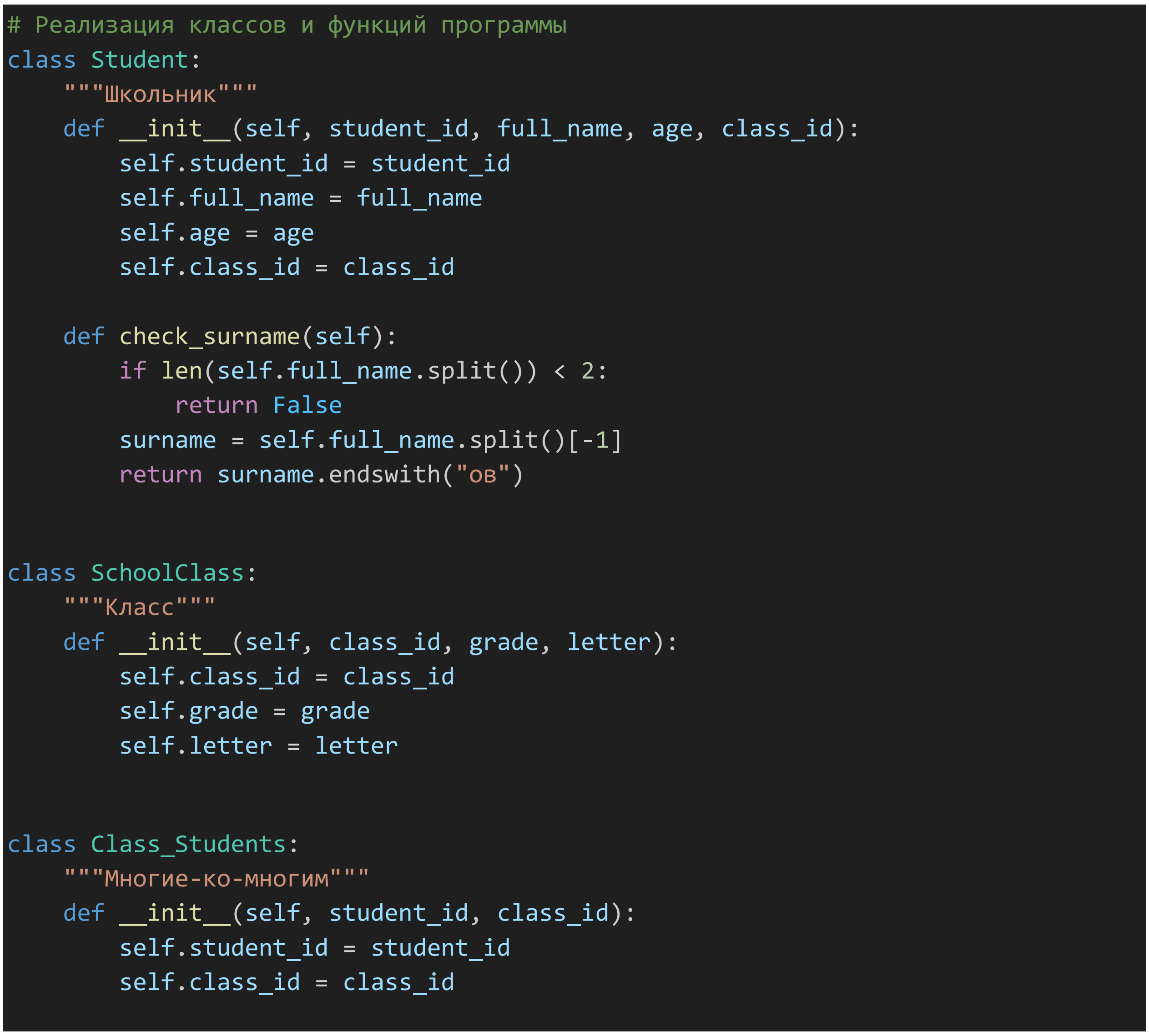
Результаты выполнения



```
PS D:\фотографии для видео\ВУЗ\программирование\rk2> python test.py
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
PS D:\фотографии для видео\ВУЗ\программирование\rk2>
```

Текст программ



```python
# Реализация классов и функций программы
class Student:
    """Школьник"""
    def __init__(self, student_id, full_name, age, class_id):
        self.student_id = student_id
        self.full_name = full_name
        self.age = age
        self.class_id = class_id

    def check_surname(self):
        if len(self.full_name.split()) < 2:
            return False
        surname = self.full_name.split()[-1]
        return surname.endswith("ов")


class SchoolClass:
    """Класс"""
    def __init__(self, class_id, grade, letter):
        self.class_id = class_id
        self.grade = grade
        self.letter = letter


class Class_Students:
    """Многие-ко-многим"""
    def __init__(self, student_id, class_id):
        self.student_id = student_id
        self.class_id = class_id
```

```python
# Функции

def get_class_name(school_classes, class_id):
    for school_class in school_classes:
        if school_class.class_id == class_id:
            return f"{school_class.grade}{school_class.letter}"
    return "NS"


def find_by_surname(students, school_classes):
    return [
        {
            "full_name": student.full_name,
            "age": student.age,
            "class": get_class_name(school_classes, student.class_id),
        }
        for student in students
        if student.check_surname()
    ]


def class_by_average_age(students, school_classes):
    result = []
    for school_class in school_classes:
        ages = [student.age for student in students if student.class_id ==
school_class.class_id]
        avg_age = sum(ages) / len(ages) if ages else 0
        result.append({"class": f"{school_class.grade}{school_class.letter}",
"average_age": avg_age})
    return sorted(result, key=lambda x: x["average_age"])


def find_classes_with_a(students, school_classes, relations):
    result = []
    for school_class in school_classes:
        if school_class.letter == "A":
            class_students = [
                {
                    "full_name": student.full_name,
                    "age": student.age,
                }
                for relation in relations
                if relation.class_id == school_class.class_id
                for student in students
                if student.student_id == relation.student_id
            ]
            result.append({"class": f"{school_class.grade}{school_class.letter}",
"students": class_students})
    return result
```

test.py

```python
import unittest
from rk2 import Student, SchoolClass, Class_Students, find_by_surname,
class_by_average_age, find_classes_with_a

class TestSchoolProgram(unittest.TestCase):
    def setUp(self):
        self.students = [
            Student(1, "Иван Иванов", 11, 1),
            Student(2, "Мария Петрова", 12, 1),
            Student(3, "Цекарь Эйсап", 12, 2),
            Student(4, "Анна Кузнецова", 13, 2),
            Student(5, "Игорь Гофман", 14, 3),
            Student(6, "Спартак Бендеров", 17, 4),
            Student(7, "Дмитрий Соколовский", 13, 2),
        ]
        self.school_classes = [
            SchoolClass(1, 5, "А"),
            SchoolClass(2, 7, "Б"),
            SchoolClass(3, 6, "В"),
            SchoolClass(4, 10, "А"),
        ]
        self.relations = [
            Class_Students(1, 1),
            Class_Students(2, 1),
            Class_Students(3, 2),
            Class_Students(4, 2),
            Class_Students(5, 3),
            Class_Students(6, 4),
            Class_Students(7, 2),
        ]

    def test_find_by_surname(self):
        result = find_by_surname(self.students, self.school_classes)
        self.assertEqual(len(result), 2)
        self.assertEqual(result[0]["full_name"], "Иван Иванов")
        self.assertIn("Иван Иванов", [student["full_name"] for student in
result])

    def test_class_by_average_age(self):
        result = class_by_average_age(self.students, self.school_classes)
        self.assertEqual(result[0]["class"], "5A")
        self.assertAlmostEqual(result[0]["average_age"], 11.5)
        self.assertGreater(result[-1]["average_age"], result[0]["average_age"])

    def test_find_classes_with_a(self):
        result = find_classes_with_a(self.students, self.school_classes,
self.relations)
        self.assertEqual(len(result), 2)
        self.assertEqual(result[0]["class"], "5A")
        self.assertTrue(any(student["full_name"] == "Иван Иванов" for student in 
result[0]["students"]))

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