

текст программы

рефракторинг кода РК1:

```
from operator import itemgetter

class Student:
    """Школьник"""
    def __init__(self, id, name, grade, class_id):
        self.id = id
        self.name = name
        self.grade = grade
        self.class_id = class_id

class SchoolClass:
    """Класс"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class StudentClass:
    """'Школьники класса' для реализации связи многие-ко-многим"""
    def __init__(self, class_id, student_id):
        self.class_id = class_id
        self.student_id = student_id

def get_one_to_many(students, classes):
    """Соединение данных один-ко-многим"""
    return [(s.name, s.grade, c.name)
            for c in classes
            for s in students
            if s.class_id == c.id]

def get_many_to_many(students, classes, students_classes):
    """Соединение данных многие-ко-многим"""
    many_to_many_temp = [(c.name, sc.class_id, sc.student_id)
                          for c in classes
                          for sc in students_classes
                          if c.id == sc.class_id]
```

```
return [(s.name, s.grade, class_name)
for class_name, class_id, student_id in many_to_many_temp
for s in students if s.id == student_id]
```

```
def task_a1(one_to_many):
    """Задание A1: Список всех школьников и их классов, отсортированный по классам"""
    return sorted(one_to_many, key=itemgetter(2))
```

```
def task_a2(one_to_many, classes):
    """Задание A2: Список классов с суммарными оценками школьников"""
    res_a2_unsorted = []
    for c in classes:
        class_students = list(filter(lambda i: i[2] == c.name, one_to_many))
        if class_students:
            total_grades = sum([grade for _, grade, _ in class_students])
            res_a2_unsorted.append((c.name, total_grades))
    return sorted(res_a2_unsorted, key=itemgetter(1), reverse=True)
```

```
def task_a3(many_to_many, classes):
    """Задание A3: Классы с названием "класс" и их школьники"""
    res_a3 = {}
    for c in classes:
        if 'класс' in c.name.lower():
            class_students = list(filter(lambda i: i[2] == c.name, many_to_many))
            if class_students: # Добавляем проверку, чтобы исключить пустые классы
                student_names = [name for name, _, _ in class_students]
                res_a3[c.name] = student_names
    return res_a3
```

```
def main():
    """Основная функция"""
    classes = [
        SchoolClass(1, 'Класс А'),
        SchoolClass(2, 'Класс Б'),
        SchoolClass(3, 'Класс В'),
        SchoolClass(4, 'Математический класс'),
        SchoolClass(5, 'Физический класс'),
    ]
```

```
students = [
    Student(1, 'Алексей', 90, 1),
    Student(2, 'Мария', 85, 2),
    Student(3, 'Сергей', 92, 3),
    Student(4, 'Николай', 88, 3),
```

```
Student(5, 'Анна', 95, 1),  
]
```

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

```
one_to_many = get_one_to_many(students, classes)  
many_to_many = get_many_to_many(students, classes, students_classes)
```

```
print('Задание A1')  
print(task_a1(one_to_many))
```

```
print('\nЗадание A2')  
print(task_a2(one_to_many, classes))
```

```
print('\nЗадание A3')  
print(task_a3(many_to_many, classes))
```

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

Тесты:

```
import unittest  
from main_program import Student, SchoolClass, StudentClass, get_one_to_many, get_many_to_many, task_a1,  
task_a2, task_a3
```

```
class TestControlWork(unittest.TestCase):
```

```
    def setUp(self):  
        self.students = [  
            Student(1, 'Алексей', 90, 1),  
            Student(2, 'Мария', 85, 2),  
            Student(3, 'Сергей', 92, 3),  
            Student(4, 'Николай', 88, 3),  
            Student(5, 'Анна', 95, 1),
```

```
]
```

```
self.classes = [  
    SchoolClass(1, 'Класс А'),  
    SchoolClass(2, 'Класс Б'),  
    SchoolClass(3, 'Класс В'),  
    SchoolClass(4, 'Математический класс'),  
    SchoolClass(5, 'Физический класс'),  
]
```

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

```
self.one_to_many = get_one_to_many(self.students, self.classes)  
self.many_to_many = get_many_to_many(self.students, self.classes, self.students_classes)
```

```
def test_task_a1(self):  
    expected = [  
        ('Алексей', 90, 'Класс А'),  
        ('Анна', 95, 'Класс А'),  
        ('Мария', 85, 'Класс Б'),  
        ('Сергей', 92, 'Класс В'),  
        ('Николай', 88, 'Класс В'),  
    ]  
    self.assertEqual(task_a1(self.one_to_many), expected)
```

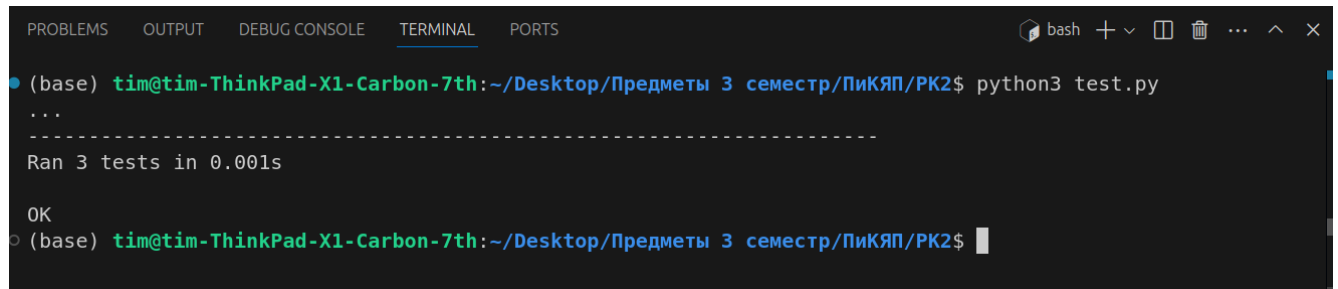
```
def test_task_a2(self):  
    expected = [  
        ('Класс А', 185),  
        ('Класс В', 180),  
        ('Класс Б', 85),  
    ]  
    self.assertEqual(task_a2(self.one_to_many, self.classes), expected)
```

```
def test_task_a3(self):  
    expected = {  
        'Класс А': ['Алексей', 'Анна'],  
        'Класс Б': ['Мария'],  
        'Класс В': ['Сергей', 'Николай'],  
    }  
    self.assertEqual(task_a3(self.many_to_many, self.classes), expected)
```

```
if __name__ == '__main__':
```

```
unittest.main()
```

Результат выполнения программы



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash + - [ ] [X] ... ^ X
(base) tim@tim-ThinkPad-X1-Carbon-7th:~/Desktop/Предметы 3 семестр/ПикЯП/РК2$ python3 test.py
...
-----
Ran 3 tests in 0.001s

OK
(base) tim@tim-ThinkPad-X1-Carbon-7th:~/Desktop/Предметы 3 семестр/ПикЯП/РК2$
```

Дата разработки: 18.12.2024
Подпись: