Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»
Отчет по рубежному контролю №2

Выполнил:
студент группы ИУ5-31Б
Абуховский Иван
Александрович

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий
Евгеньевич

Подпись:_____

Дата:_____

Москва, 2021 г.

Рубежный контроль №2
Задание

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

AbukhovskiyRK1IU5_31B.py

```
from operator import itemgetter

class Student:
    """Студент"""
    def __init__(self, id, surname, dolgi, group_id):
        self.id = id
        self.surname = surname
        self.dolgi = dolgi
        self.group_id = group_id

class Group:
    """Группа"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class StudentGroup:
    """
    'Студенты группы' для реализации
    связи многие-ко-многим
    """
    def __init__(self, group_id, student_id):
        self.group_id = group_id
        self.student_id = student_id

# Группы
groups = [
    Group(1, 'ИУ5-31Б'),
    Group(2, 'ИУ6-12Б'),
    Group(3, 'ИУ7-53Б'),
    Group(4, 'ИУ8-34Б'),
    Group(5, 'ИУ9-15Б'),

    Group(11, 'МТ5-11'),
    Group(22, 'РК5-32'),
    Group(33, 'СМ5-13'),
    Group(44, 'Э5-54'),
    Group(55, 'ОЭП5-35'),
]

# Студенты
students = [
    Student(1, 'Абуховский', 25, 1),
    Student(2, 'Черноморец', 47, 2),
    Student(3, 'Шагиахметов', 39, 3),
    Student(4, 'Стельмах', 81, 4),
```

```
    Student(5, 'Сыса', 17, 5),
    Student(6, 'Слепченкова', 43, 11),
    Student(7, 'Кузьмин', 31, 22),
    Student(8, 'Соколов', 29, 33),
    Student(9, 'Заточен', 48, 44),
    Student(10, 'Проценко', 33, 55),
    Student(11, 'Калинников', 21, 1),
    Student(12, 'Милевич', 21, 2),
    Student(13, 'Слоква', 65, 3),
    Student(14, 'Барабанщиков', 15, 4),
    Student(15, 'Акулова', 83, 5),
    Student(16, 'Ашуров', 12, 11),
    Student(17, 'Бекетов', 14, 22),
    Student(18, 'Поляков', 11, 33),
    Student(19, 'Нигматуллин', 10, 44),
    Student(20, 'Собакевич', 10, 55),
    Student(21, 'Ахтамбаев', 32, 3),
    Student(22, 'Цуприков', 15, 33),
]

students_groups = [
    StudentGroup(1,1),
    StudentGroup(1,11),
    StudentGroup(2,2),
    StudentGroup(2,12),
    StudentGroup(3,3),
    StudentGroup(3,13),
    StudentGroup(4,4),
    StudentGroup(4,13),
    StudentGroup(5,5),
    StudentGroup(5,15),
    StudentGroup(11,6),
    StudentGroup(11,16),
    StudentGroup(22,7),
    StudentGroup(22,17),
    StudentGroup(33,8),
    StudentGroup(33,18),
    StudentGroup(44,9),
    StudentGroup(44,19),
    StudentGroup(55,10),
    StudentGroup(55,20),
    StudentGroup(3,21),
    StudentGroup(33,22),
]

def main():
    """Основная функция"""

    # Соединение данных один-ко-многим
    one_to_many = [(s.surname, s.dolgi, g.name)
        for g in groups
        for s in students
        if s.group_id==g.id]

    # Соединение данных многие-ко-многим
    many_to_many_temp = [(g.name, sg.group_id, sg.student_id)
        for g in groups
        for sg in students_groups
        if g.id==sg.group_id]

    many_to_many = [(s.surname, s.dolgi, group_name)
        for group_name, group_id, student_id in many_to_many_temp
        for s in students if s.id==student_id]

    print('Задание А1')
```

```
    res_11 = sorted(one_to_many, key=itemgetter(2))
    print(res_11)

    print('\nЗадание А2')
    res_12_unsorted = []
    # Перебираем все группы
    for g in groups:
        # Список студентов группы
        g_students = list(filter(lambda i: i[2]==g.name, one_to_many))
        # Если группа не пустая
        if len(g_students) > 0:
            # Долги студентов группы
            g_dolgi = [dolgi for _,dolgi,_ in g_students]
            # Суммарное количество долгов студентов группы
            g_dolgi_sum = sum(g_dolgi)
            res_12_unsorted.append((g.name, g_dolgi_sum))

    # Сортировка по суммарной зарплате
    res_12 = sorted(res_12_unsorted, key=itemgetter(1), reverse=True)
    print(res_12)

    print('\nЗадание А3')
    res_13 = {}
    # Перебираем все группы
    for g in groups:
        if 'Б' in g.name:
            # Список студентов группы
            g_students = list(filter(lambda i: i[2]==g.name, many_to_many))
            # Только ФИО сотрудников
            g_students_surnames = [x for x,_,_ in g_students]
            # Добавляем результат в словарь
            # ключ - отдел, значение - список фамилий
            res_13[g.name] = g_students_surnames

    print(res_13)

if __name__ == '__main__':

      main()
```

```
    Pup(3, 'Зелинский', 10000, 2),
    Pup(4, 'Бондаренко', 10000, 3),
    Pup(5, 'Смыслов', 30000, 3),
]

pups_clss = [
    PupCls(1, 1),
    PupCls(2, 2),
    PupCls(2, 3),
    PupCls(3, 4),
    PupCls(3, 5),

    PupCls(11, 1),
    PupCls(22, 2),
    PupCls(22, 3),
    PupCls(33, 4),
    PupCls(33, 5),
]


def task_a1(one_to_many):
    one_to_many = one_to_many
    return [i for i in sorted(one_to_many, key=itemgetter(2))]


def task_a2(one_to_many):
    res_2_unsorted = []
    # Перебираем все классы
    for c in clss:
        # Список школьников класса
        c_pups = list(filter(lambda i: i[2] == c.name, one_to_many))
        # Если класс не пустой
        if len(c_pups) > 0:
            # Долги за обучение школьников класса
            c_dbts = [dbt for _, dbt, _ in c_pups]
            # Суммарный долг школьников класса
            c_dbts_sum = sum(c_dbts)
            res_2_unsorted.append((c.name, c_dbts_sum))

    # Сортировка по суммарному долгу
    res_2 = sorted(res_2_unsorted, key=itemgetter(1), reverse=True)
    return res_2


def task_a3(many_to_many):
    res_3 = {}
    # Перебираем все классы
    for c in clss:
        if 'В' in c.name:
            # Список школьников класса
            c_pups = list(filter(lambda i: i[2] == c.name, many_to_many))
            # Только ФИО школьников
            c_pups_names = [x for x, _, _ in c_pups]
            # Добавляем результат в словарь
            # ключ - класс, значение - список фамилий
            res_3[c.name] = c_pups_names
    return res_3


def main():

    # Соединение данных один-ко-многим
    one_to_many = [(p.fio, p.dbt, c.name)
```

```
                    for c in clss
                    for p in pups
                    if p.cls_id == c.id]

    # Соединение данных многие-ко-многим
    many_to_many_temp = [(c.name, pc.cls_id, pc.pup_id)
                         for c in clss
                         for pc in pups_clss
                         if c.id == pc.cls_id]

    many_to_many = [(p.fio, p.dbt, cls_name)
                    for cls_name, cls_id, pup_id in many_to_many_temp
                    for p in pups if p.id == pup_id]

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

    print('\nЗадание A2')
    print(task_a2(one_to_many))

    print('\nЗадание A3')
    print(task_a3(many_to_many))


if_name_== '_main__':
    main()
```

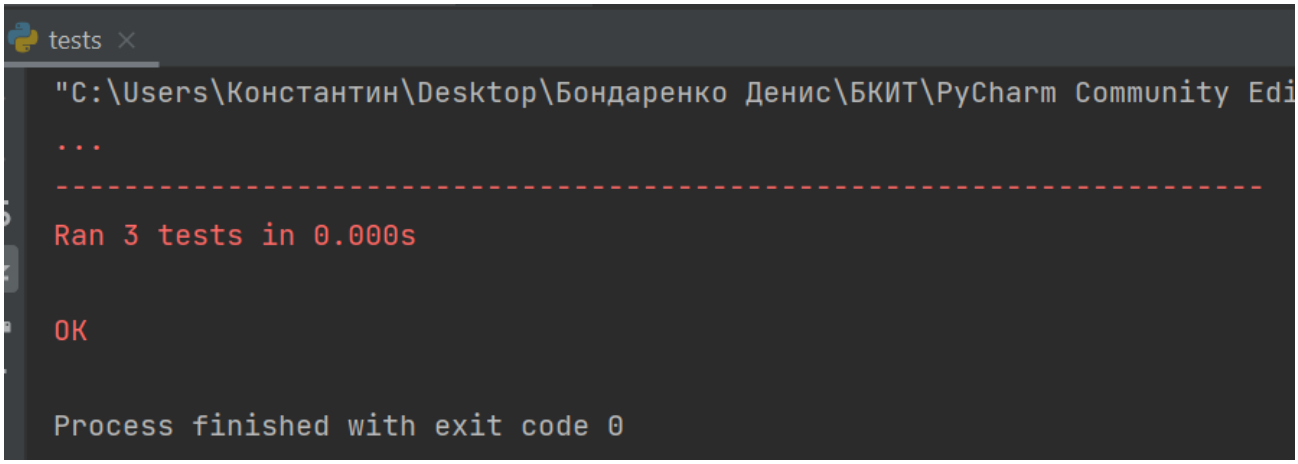
tests.py

```
from School2 import *
import unittest


class Test(unittest.TestCase):

    def setUp(self):
        self.clss = [
                    Cls(1, '11А'),
                    Cls(2, '7Б'),
                    Cls(3, '5В'),

                    Cls(11, '11Б'),
                    Cls(22, '7В'),
                    Cls(33, '5А'),
                    ]
        self.pups = [
                    Pup(1, 'Абуховский', 5000, 1),
                    Pup(2, 'Рыжкова', 0, 2),
                    Pup(3, 'Зелинский', 10000, 2),
                    Pup(4, 'Бондаренко', 10000, 3),
                    Pup(5, 'Смыслов', 30000, 3),
                    ]
        self.pups_clss = [
                          PupCls(1, 1),
                          PupCls(2, 2),
                          PupCls(2, 3),
                          PupCls(3, 4),
                          PupCls(3, 5),

                          PupCls(11, 1),
                          PupCls(22, 2),
                          PupCls(22, 3),
```

```
                        PupCls(33, 4),
                        PupCls(33, 5),
                        ]

        self.one_to_many = [(p.fio, p.dbt, c.name)
                            for c in clss
                            for p in pups
                            if p.cls_id == c.id]

        self.many_to_many_temp = [(c.name, pc.cls_id, pc.pup_id)
                                  for c in clss
                                  for pc in pups_clss
                                  if c.id == pc.cls_id]

        self.many_to_many = [(p.fio, p.dbt, cls_name)
                             for cls_name, cls_id, pup_id in
self.many_to_many_temp
                             for p in pups if p.id == pup_id]

    def test_task_a1(self):
        prediction = [('Абуховский', 5000, '11А'), ('Бондаренко', 10000, '5В'),
('Смыслов', 30000, '5В'),
                      ('Рыжкова', 0, '7Б'), ('Зелинский', 10000, '7Б')]
        self.assertEqual(task_a1(self.one_to_many), prediction)

    def test_task_a2(self):
        prediction = [('5В', 40000), ('7Б', 10000), ('11А', 5000)]
        self.assertEqual(task_a2(self.one_to_many), prediction)

    def test_task_a3(self):
        prediction = {'7Б': ['Рыжкова', 'Зелинский'], '11Б': ['Абуховский']}
        self.assertEqual(task_a3(self.many_to_many), prediction)


if_name_== '_main_':
    unittest.main()
```

Результаты выполнения программы

Программа была отредактирована так, чтобы результаты ее выполнения не изменились, но ее можно было бы модульно протестировать.



Тестирование показывает, что программа работает так, как и предполагается.

Если внести изменения в ожидаемый результат, то результат тестирования укажет на несоответствие полученного результата ожидаемому.