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
Бондаренко Денис
Константинович

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

School2.py

```
from operator import itemgetter


class Pup:
    """Школьник"""

    def __init__(self, id, fio, dbt, cls_id):
        self.id = id
        self.fio = fio
        self.dbt = dbt
        self.cls_id = cls_id


class Cls:
    """Класс"""

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


class PupCls:
    """
    'Школьники класса' для реализации
    связи многие-ко-многим
    """

    def __init__(self, cls_id, pup_id):
        self.cls_id = cls_id
        self.pup_id = pup_id


# Классы
clss = [
    Cls(1, '11А'),
    Cls(2, '7Б'),
    Cls(3, '5В'),

    Cls(11, '11Б'),
    Cls(22, '7В'),
    Cls(33, '5А'),
]

# Школьники
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),
]

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 'Б' 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)
```

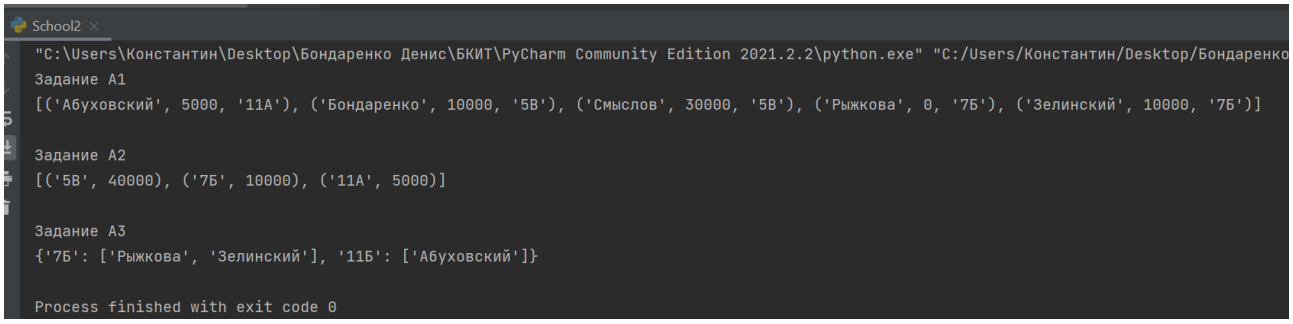
```
                  for c in clss
                  for p in pups
                  if p.cls_id == c.id]

    # 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

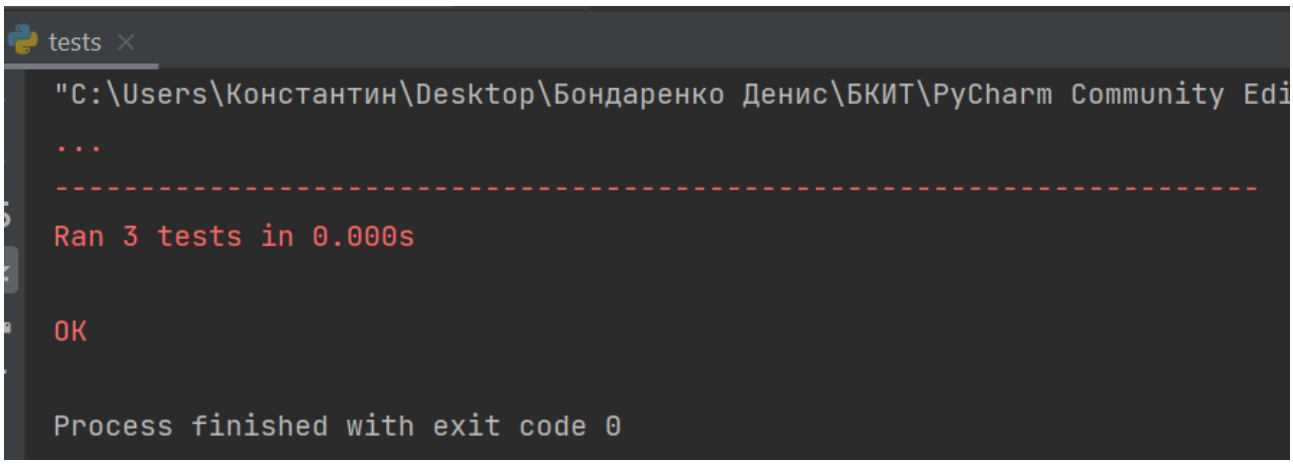
```
from School2 import *
import unittest


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.