Московский государственный технический университет имени Н. Э. Баумана

Факультет Радиотехнический

Кафедра ИУ5 Системы обработки информации и управления

Отчет по рубежному контролю №2 по курсу Базовые компоненты интернет-технологий

8
(количество листов)

Вариант № 17 (Е)

Исполнитель
студент группы РТ5-31б _____ Павлов С.Д.
“ 14 ” декабря 2021 г.

Проверил
Доцент кафедры ИУ5 _____ Гапанюк Ю.Е.
“___” _____ 2021 г.

Москва, 2021 г.

Задание

Условия рубежного контроля №2 по курсу БКИТ

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Провести рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайть модульные тесты с применением TDD - фреймворка (3 теста)

Текст программы

Файл Rk2_Classes.py:

```
from unittest import TestCase
from random import randint
from main import *
from os import system

#Класс дирижера:
class Conductor:
    conductors_number = 0 #Счетчик для id
    def __init__(self, Orchestra_id=None, Name="", Sername="", Middle_name="", Salary=None,
Orchestra_name=""):
        self.conductor_id = Conductor.conductors_number
        Conductor.conductors_number+=1
        self.orchestra_id = Orchestra_id
        self.name         = Name
        self.sername      = Sername
        self.middle_name  = Middle_name
        self.salary       = Salary

    #Краткий вывод экземпляра:
    def show(self, Logic=True):
        line = self.name + " " + self.sername
        if Logic == True: print(line + " (id = " + str(self.conductor_id) \
        + ", оркестр_id = " + str(self.orchestra_id) + ")", end="")
        return line

#Класс оркестра:
class Orchestra:
    orchestras_number = 0 #Счетчик для id
    def __init__(self, Name=""):
        self.orchestra_id = Orchestra.orchestras_number
        Orchestra.orchestras_number+=1
        self.name = Name

    #Краткий вывод экземпляра:
```

```
    def show(self, Logic=True):
        line = "\"" +self.name + "\""
        if Logic == True: print(line + " (id = " + str(self.orchestra_id) + ")", end="")
        return line

#Класс дирижеров оркестров (для реализации связей и хранения данных):
class Storage:
    orchestras_dict  = {} #Словарь относительно оркестров
    orchestras_list  = [] #Списки оркестров и дирижеров
    conductors_list  = [] #

    def __init__(self):
        self.orchestras_dict  = {}
        self.orchestras_list  = []
        self.conductors_list  = []
        Orchestra.orchestras_number = 0
        Conductor.conductors_number = 0

    #Добавление новых дирижеров:
    def add_conductor(self, Conductors, Orchestra_id):
        #Поиск соответсвующего оркестра
        conductors_of_orchestra = self.orchestras_dict.get(Orchestra_id, None)
        if conductors_of_orchestra != None:
            #Если такой имеется и соблюден синтаксис добавляем необходимых дирижеров
            if isinstance(Conductors, list) == True:
                for i in range(len(Conductors)):
                    #в лист связей:
                    conductors_of_orchestra.append(Conductors[i].conductor_id)
                    self.conductors_list.extend(Conductors) #в хранилище дирижеров
            else: print("Ошибка синтаксиса!")
        else: print("Отсутсвует данный оркестр!")

    #Добавление новых оркестров:
    def add_orchestra(self, Orchestras):
        if isinstance(Orchestras, list) == True:
            for i in range(len(Orchestras)):
                self.orchestras_dict[Orchestras[i].orchestra_id] = [] #в лист связей
            self.orchestras_list.extend(Orchestras) #в хранилище оркестров
        else: print("Ошибка синтаксиса!")

    #Возвращение дирижера по id:
    def return_conductor_by_id(self, Conductor_id):
        try: return self.conductors_list[Conductor_id]
        except: return None

    #Возвращение id оркестра по его имени:
    def find_orcestra_by_name(Storage, Name):
        for i in range(len(Storage.orchestras_list)):
            if Storage.orchestras_list[i].name == Name: return i
        return None

    #Создание отсортированного по возрастанию списка средних зарплат в оркестрах:
    def count_average_salary(self):
        sallary_range_list = []
        for i in range(len(self.orchestras_dict)):
            summ = 0
            for i2 in range(len(self.orchestras_dict[i])):
                sallary = self.conductors_list[self.orchestras_dict[i][i2]].salary
                if sallary!= None:
                    summ += sallary #Сумма месячных зарплат в этом оркестре
            length = len(self.orchestras_dict[i])
```

```
            if length>0:
                Averange = round(summ/length, 2) #Находим среднее
            else:
                Averange = 0
            #Добавляем его соответсвенно с id оркестра
            sallary_range_list.append([Averange, i])
        sallary_range_list.sort()
        return sallary_range_list

#================ТЕСТИРОВАНИЕ================
#Класс, создающий случайные данные для испытаний:
class Init_Random_Storage(Storage):
    def init(self, number_conductores, number_orchestras, length_a,
             length_b, salary_a, salary_b):
        for i in range(number_orchestras):
            self.add_orchestra([Orchestra(self.get_rand_name(length_a, length_b))])

        for i in range(number_conductores):
            R = randint(0, number_orchestras-1)
                self.add_conductor([Conductor(R, self.get_rand_name(length_a, length_b),
                                              self.get_rand_name(length_a, length_b),
                                              self.get_rand_name(length_a, length_b),
                                              randint(salary_a, salary_b))], R
                                              )

    def get_rand_name(self, legth_a,length_b):
        Name = ""
        letters = "абвгдежзийкльмнопрстуфцчшщъыьэюяАБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ"
        length = randint(legth_a, length_b)
        for i in range(length):
            Name += letters[randint(0, 63)]
        return Name

#Проверка функции подсчета среднего заработка:
class Test_of_count_average_salary(TestCase):
    def test_check(self, times=15):
        print("\nПроверка функции подсчета среднего заработка: ", end="")
        for i in range(times):
            S = Init_Random_Storage()
            S.init(10, 10, 10, 15, 10000, 1000000)
            awarange_salary = S.count_average_salary()

            #Проверка порядка:
            m=0
            for i in awarange_salary:
                Test_of_count_average_salary.assertTrue(self, i[0]>=m)
                a=m

            #Проверка наличия:
            for i in range(0, len(S.orchestras_dict)-1):
                summ = 0;
                for i2 in range(len(S.orchestras_dict[i])):
                    salary = S.conductors_list[S.orchestras_dict[i][i2]].salary
                    if salary != None:
                        summ += salary
                if len(S.orchestras_dict[i])>0:
                    awarange = round(summ/len(S.orchestras_dict[i]), 2)
                else:
                    awarange = 0
                Test_of_count_average_salary.assertIn(self, [awarange, i], awarange_salary)
        print("Ok",end="")
```

```
#Проверка функции вывода среднего заработка:
class Test_of_show_middle_salary_list(TestCase):
    def test_check(self):
        print("\nПроверка функции вывода среднего заработка: ", end="")
        S = Init_Random_Storage()
        S.init(10, 10, 10, 15, 10000, 1000000)
        #Считается, что эта функция уже проверена:
        average_salaries_list = S.count_average_salary()

        num=0
        for i in show_middle_salary_list(S.orchestras_list, average_salaries_list):
            Test_of_show_middle_salary_list.assertTrue(self,
                                            i["срзп"] == average_salaries_list[num][0])

            Test_of_show_middle_salary_list.assertTrue(self, i["id"] == "(id = " +
                                            str(average_salaries_list[num][1]) + ")")

            Test_of_show_middle_salary_list.assertTrue(self,
              i["название"] == S.orchestras_list[average_salaries_list[num][1]].show(False))

            num+=1
        print("Ok", end="")

#Проверка функции, возвращающей дирижеров из списка:
class Test_of_return_conductors(TestCase):
    def test_check(self):
        print("\nПроверка функции, возвращающей дирижеров из списка: ", end="")
        S = Init_Random_Storage()
        S.init(10, 10, 10, 15, 10000, 1000000)
        ids_list = []

        #Получаем все id дирижеров, проходя по списку:
        for i in S.conductors_list:
            ids_list.append(i.conductor_id)

        #Проверяем их наличие:
        for i in return_conductors(S, ids_list):
                Test_of_return_conductors.assertIn(self, i, S.conductors_list)
        print("Ok", end="")

#Проверка функции поиска слова в оркестре:
class Test_of_word_in_orchestra_search(TestCase):
    def test_check(self, times=100):
        print("\nПроверка функции поиска слова в оркестре: ", end="")
        for i in range(times):
            S = Init_Random_Storage()
            S.init(10, 10, 5, 8, 10000, 1000000)
            word =  S.get_rand_name(1, 1)
            #Проверка правильности:
            num1 = 0
            for i in  word_in_orchestra_search(S.orchestras_list, S.conductors_list,
                                            S.orchestras_dict, word):
                Test_of_word_in_orchestra_search.assertIn(self, word, i.name)
                num1+=1

            #Проверка того, что найдено все:
            num2 = 0
            for i in range(len(S.orchestras_list)):
                if word in S.orchestras_list[i].name:
                    num2+=1
            Test_of_word_in_orchestra_search.assertTrue(self, num1==num2)
```

```
        print("Ok", end="")

#Проверка функции поиска дирижера по первой букве:
class Test_of_check_conductures_first_letter(TestCase):
    def test_check(self, times=10):
        print("\nПроверка функции поиска дирижера по первой букве: ", end="")
        for i in range(times):
            S = Init_Random_Storage()
            S.init(10, 10, 10, 15, 10000, 1000000)
            letter =  S.get_rand_name(1, 1)

            #Проверка правильности:
            num1 = 0
            for i in check_conductures_first_letter(S.conductors_list, S.orchestras_list,
                                                    letter):
                Test_of_check_conductures_first_letter.assertTrue(self,
                                                                  i["фамилия"][0]==letter)
                num1+=1

            #Проверка того, что найдено все:
            num2 = 0
            for i in S.conductors_list:
                if i.sername[0] == letter:
                    num2 += 1
            Test_of_check_conductures_first_letter.assertTrue(self, num1==num2)


 print("Ok", end="")
```

Файл main.py:

```
from Rk2_Classes import *
from os import system
from unittest import main

def word_in_orchestra_search(orchestras_list, conductors_list, orchestras_dict, word):
    for i in range(len(orchestras_list)):
        if word in orchestras_list[i].name:
            conductors_list.extend(orchestras_dict[i])
            yield orchestras_list[i]

def return_conductors(storage, conductors_list):
    for i in range(len(conductors_list)):
        yield storage.return_conductor_by_id(conductors_list[i])

def show_middle_salary_list(orcestra_list, average_salaries_list):
    for i in range(len(average_salaries_list)):
        orcestra_id = average_salaries_list[i][1]
        yield {"название": orcestra_list[orcestra_id].show(False),
               "id": "(id = " + str(orcestra_id) + ")",
               "срзп": average_salaries_list[i][0]
               }

def check_conductures_first_letter(conductors_list, orcestra_list, letter):
    for i in range(len(conductors_list)):
        if letter == (conductors_list[i].sername)[0]:
            orcestra_id = conductors_list[i].orchestra_id
            yield {"фио": conductors_list[i].show(False),
                   "id": "(id = " + str(conductors_list[i].conductor_id) + ")",
```

```
                    "оркестр": orcestra_list[orcestra_id].name,
                    "фамилия": conductors_list[i].sername
                    }

def main1():
    S = Storage()

    #Создаем оркестры:
    S.add_orchestra([Orchestra("Малый симфонический"), Orchestra("Практикантов (малый)"),
                     Orchestra("Большой симфонический"), Orchestra("Очень малый малый")])

    new_orchestra = Orchestra("Не знаю, как назвать :<")
    S.add_orchestra([new_orchestra])

    #Создаем дирижеров:
    S.add_conductor([Conductor(0, "Иван",    "Крылов", "Матвеевич",  250000),
                     Conductor(0, "Григорий", "Авдеев", "Артемович",    300000)], 0)

    S.add_conductor([Conductor(1, "Troll",   "Face",   "Memow",      15000),
                     Conductor(1, "V", "Анонимный",      "Anonimous",    1830000)], 1)

    S.add_conductor([Conductor(2, "Василий", "Шуткин", "Тотсамович", 257000),
                     Conductor(2, "Товарищ",  "Майор",  "Вездесущевич", 100)], 2)

    S.add_conductor([Conductor(S.find_orcestra_by_name("Очень малый малый"), "Антон",
                    "Антонов", "Михеевич", 248000)],
                     S.find_orcestra_by_name("Очень малый малый"))

    S.add_conductor([Conductor(S.find_orcestra_by_name("Очень малый малый"), "Рептилоид",
                    "Мировопорядков", "Иллюминатович", 248000)],
                     S.find_orcestra_by_name("Не знаю, как назвать :<"))

    #================================[E1]================================#
    workers_list = [] #Лист дирижеров оркестров, удовлетворяющих требованию

    #Выведем все оркестры, в названия которых входит слово "малый"(с учетом регистра)
    print("Оркестры, содержащие в названии слово \"малый\" (с учетом регистра):")
    for i in word_in_orchestra_search(S.orchestras_list, workers_list,
                                      S.orchestras_dict, "малый"):
        i.show(); print("; ", end ="")

    #И всех дирижеров этих оркестров:
    print("\n\nДирижеры этих оркестров:")
    for i in return_conductors(S, workers_list):
            i.show(); print("; ", end="")

    #================================[E2]================================#
    #Рассчитаем среднюю зарплату и занесем во вложенный список:
    print("\n\n\nОркестры по возрастанию средней зарплаты в них:")
    average_salaries_list = S.count_average_salary()

    #Выведем его:
    for i in show_middle_salary_list(S.orchestras_list, average_salaries_list):
       print("%-26s%-9s%-3s%6d%7s" % (i["название"], i["id"], "->", i["срзп"], "рублей"))

    #================================[E3]================================#
    #Выведем список всех дирижеров, чья фамилия начинается на А и названия их отделов:
    print("\n\n\nСписок всех дирижеров, чья фамилия начинается на А \
            и их названия оркестров:")
    for i in check_conductures_first_letter(S.conductors_list, S.orchestras_list, "А"):
        print("%-16s%-9s%-9s%6s" % (i["фио"], i["id"], "оркестр:", i["оркестр"]))
```

```
if __name__ == "__main__":
    #main1()
    #system("pause")
    main()
```

Результат выполнения программы: