Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехнический»

Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

Выполнил:

студент группы РТ5-31Б:

Гладышев А.К.

Подпись и дата:

Проверил:

преподаватель кафедры ИУ5

Гапанюк Ю.Е.

Подпись и дата:

Москва, 2024 г.

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

```
class DispClass:

    def __init__(self, id_dispClass: int, audit_numb: int, admin_name: str, cmp_nub: int):

        self._id = id_dispClass

        self._numb = audit_numb

        self._admin = admin_name

        self._num_comp = cmp_nub


    @property

    def get_id(self) -> int:

        return self._id


    @property

    def get_aud_num(self) -> int:

        return self._numb


    @property

    def get_admin_name(self) -> str:

        return self._admin


    @property

    def get_comp_numb(self) -> int:

        return self._num_comp
```

```
class PC:
    def __init__(self, self_id: int, comp_class_id: int, disk_size: int, motherboard:
str):
        self._id = self_id
        self._auditId = comp_class_id
        self._disk_size = disk_size
        self._mother_board = motherboard

    @property
    def get_id(self) -> int:
        return self._id

    @property
    def get_audit(self) -> int:
        return self._auditId

    @property
    def get_disk_size(self) -> int:
        return self._disk_size

    @property
    def get_mother_board(self) -> str:
        return self._mother_board

class AudPC:
    def __init__(self, id_aud: int, id_pc: int):
        self._idaud = id_aud
        self._idpc = id_pc
```

```
    @property
    def get_aud_id(self) -> int:
        return self._idaud

    @property
    def get_pc_id(self) -> int:
        return self._idpc

def task1(auditories: list[DispClass], computers: list[PC]):
    answer = [(a, c) for a in auditories for c in computers if a.get_id == c.get_audit
and a.get_admin_name == "Nikita"]
    return [(a.get_admin_name, a.get_id, c.get_id, c.get_mother_board) for (a, c) in
answer]

def task2(auditories: list[DispClass], computers: list[PC]):
    avg_space = []
    for otd in auditories:
        avg_sum = sum(comp.get_disk_size for comp in computers if otd.get_id ==
comp.get_audit)
        if otd.get_comp_numb > 0:
            avg_sum /= otd.get_comp_numb
            avg_space.append((round(avg_sum, 2), otd.get_id))
    avg_space.sort(reverse=True)
    return avg_space

def task3(auditories: list[DispClass], computers: list[PC], aud_pc: list[AudPC]):
    pc_with_a = []

    for otd in auditories:
```

```
        for comp in computers:
            if otd.get_id == comp.get_audit and
comp.get_mother_board.startswith('A'):
                pc_with_a.append((comp.get_id, otd.get_id, comp.get_mother_board,
otd.get_admin_name))


    for i in aud_pc:
        if computers[i.get_pc_id].get_mother_board.startswith('A'):
            pc_with_a.append((computers[i.get_pc_id].get_id,
                              auditories[i.get_aud_id].get_id,
                              computers[i.get_pc_id].get_mother_board,
                              auditories[i.get_aud_id].get_admin_name))


    return pc_with_a


auditories = [
    DispClass(0, 203, "Nikita", 3),
    DispClass(1, 204, "Vladimir", 3),
    DispClass(2, 301, "Nikita", 1)
]


computers = [
    PC(0, 0, 500, "Asus"),
    PC(1, 0, 100, "Gigiabyte"),
    PC(2, 0, 1750, "Astra"),
    PC(3, 1, 250, "Aorus"),
    PC(4, 1, 1000, "Acer"),
    PC(5, 1, 2000, "Netak"),
    PC(6, 2, 300, "zotack")
```

```
]

auditor_Comp = [
    AudPC(0, 4),
    AudPC(1, 2),
    AudPC(0, 6)
]

print(task1(auditories, computers))
print(task2(auditories, computers))
print(task3(auditories, computers, auditor_Comp))

import unittest

class TestDispClass(unittest.TestCase):

    def setUp(self):
        self.auditories = [
            DispClass(0, 203, "Nikita", 3),
            DispClass(1, 204, "Vladimir", 3),
            DispClass(2, 301, "Nikita", 1)
        ]

        self.computers = [
            PC(0, 0, 500, "Asus"),
            PC(1, 0, 1000, "Gigabyte"),
            PC(2, 0, 1750, "Astra"),
            PC(3, 1, 250, "Aorus"),
            PC(4, 1, 1000, "Acer"),
```

```
            PC(5, 1, 2000, "Netak"),
            PC(6, 2, 300, "zotack")
        ]

        self.auditor_Comp = [
            AudPC(0, 4),
            AudPC(1, 2),
            AudPC(0, 6)
        ]

    def test_task1(self):
        result = task1(self.auditories, self.computers)
        expected = [
        ('Nikita', 0, 0, 'Asus'),
        ('Nikita', 0, 1, 'Gigabyte'),
        ('Nikita', 0, 2, 'Astra'),
        ('Nikita', 2, 6, 'zotack')
        ]
        self.assertEqual(result, expected)

    def test_task2(self):
        result = task2(self.auditories, self.computers)
        expected = [(1083.33, 1), (783.33, 0), (300.00, 2)]
        self.assertEqual(result[0], expected[0])

    def test_task3(self):
        result = task3(self.auditories, self.computers, self.auditor_Comp)
        expected = [(0, 0, 'Asus', 'Nikita'), (2, 0, 'Astra', 'Nikita')]
        self.assertEqual(result[:2], expected)
```

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

Результат выполнения

```
K2/RK2.py"
[('Nikita', 0, 0, 'Asus'), ('Nikita', 0, 1, 'Gigiabyte'), ('Nikita', 0, 2, 'Astra'),
 ('Nikita', 2, 6, 'zotack')]
[(1083.33, 1), (783.33, 0), (300.0, 2)]
[(0, 0, 'Asus', 'Nikita'), (2, 0, 'Astra', 'Nikita'), (3, 1, 'Aorus', 'Vladimir'), (
4, 1, 'Acer', 'Vladimir'), (4, 0, 'Acer', 'Nikita'), (2, 1, 'Astra', 'Vladimir')]
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```