

**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  
Вариант предметной области из РК1 №4: Файл, Каталог файлов  
Вариант запросов: Е**

Выполнил:  
студент группы РТ5-31Б:  
Гаджиев Р. К.

Проверил:  
преподаватель каф. ИУ5  
Гапанюк Ю.Е.

Москва, 2025 г.

## Описание задания

Условия рубежного контроля №2 по курсу ПИК ЯП

Рубежный контроль представляет собой разработку тестов на языке Python.

- 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

```
import unittest

class Computer:
    def __init__(self, id, model, price, display_class_id):
        self.id = id
        self.model = model
        self.price = price
        self.display_class_id = display_class_id

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

class ComputerDisplayClass:
    def __init__(self, display_class_id, computer_id):
        self.display_class_id = display_class_id
        self.computer_id = computer_id

def create_data():
    display_classes = [
        DisplayClass(1, "отдел игровых компьютеров"),
        DisplayClass(2, "отдел офисных компьютеров"),
        DisplayClass(3, "учебный класс"),
        DisplayClass(4, "графический отдел"),
        DisplayClass(5, "кафе работников"),
    ]

    computers = [
        Computer(1, "ASUS ROG Strix G15", 150000, 1),
        Computer(2, "Dell OptiPlex 3090", 45000, 2),
        Computer(3, "HP EliteBook 840", 85000, 2),
        Computer(4, "MacBook Pro M2", 210000, 4),
        Computer(5, "Lenovo ThinkPad T490", 65000, 2),
        Computer(6, "MSI Gaming GE76", 180000, 1),
```

```

    Computer(7, "Acer Aspire 5", 35000, 3),
    Computer(8, "iMac 27", 195000, 4),
    Computer(9, "Asus VivoBook", 55000, 3),
]

```

```

comp_display_classes = [
    ComputerDisplayClass(1, 1),
    ComputerDisplayClass(1, 6),
    ComputerDisplayClass(2, 2),
    ComputerDisplayClass(2, 3),
    ComputerDisplayClass(2, 5),
    ComputerDisplayClass(3, 7),
    ComputerDisplayClass(3, 9),
    ComputerDisplayClass(4, 4),
    ComputerDisplayClass(4, 8),
]

```

```

return display_classes, computers, comp_display_classes

```

```

def get_one_to_many(display_classes, computers):

```

```

    return [(c.model, c.price, dc.name)
            for dc in display_classes
            for c in computers
            if c.display_class_id == dc.id]

```

```

def get_many_to_many(display_classes, computers, comp_display_classes):

```

```

    many_to_many_temp = [(dc.name, cdc.display_class_id, cdc.computer_id)
                          for dc in display_classes
                          for cdc in comp_display_classes
                          if dc.id == cdc.display_class_id]

```

```

    return [(c.model, c.price, dc_name)
            for dc_name, dc_id, c_id in many_to_many_temp
            for c in computers
            if c.id == c_id]

```

```

def query_1(display_classes, one_to_many):

```

```

    """Дисплейные классы со словом 'отдел' в названии"""

```

```

    result = {}

```

```

    dc_filtered = [dc for dc in display_classes if 'отдел' in dc.name.lower()]

```

```

    for dc in dc_filtered:

```

```

        dc_computers = [model for model, price, dc_name in one_to_many
                        if dc_name == dc.name]

```

```

        result[dc.name] = dc_computers
    return result

def query_2(display_classes, one_to_many):
    """Дисплейные классы по средней цене компьютеров"""
    result = []
    for dc in display_classes:
        dc_computers = [(model, price) for model, price, dc_name in one_to_many
                        if dc_name == dc.name]
        if dc_computers:
            prices = [price for _, price in dc_computers]
            avg_price = round(sum(prices) / len(prices), 2)
            result.append((dc.name, avg_price))
    return sorted(result, key=lambda x: x[1])

def query_3(many_to_many):
    """Компьютеры, модель которых начинается на 'A'"""
    result = {}
    computers_starting_with_a = set([model for model, price, dc_name in many_to_many
                                     if model and model[0].upper() == 'A'])

    for computer_model in computers_starting_with_a:
        computer_dc = [dc_name for model, price, dc_name in many_to_many
                       if model == computer_model]
        unique_dc = list(set(computer_dc))
        result[computer_model] = unique_dc
    return result

class TestComputerSystem(unittest.TestCase):
    def setUp(self):
        self.display_classes, self.computers, self.comp_display_classes = create_data()
        self.one_to_many = get_one_to_many(self.display_classes, self.computers)
        self.many_to_many = get_many_to_many(self.display_classes, self.computers,
        self.comp_display_classes)

    def test_query_1(self):
        """Тест: Дисплейные классы со словом 'отдел' в названии"""
        result = query_1(self.display_classes, self.one_to_many)
        self.assertIsInstance(result, dict)
        self.assertEqual(len(result), 3)

        self.assertIn("отдел игровых компьютеров", result)
        self.assertIn("отдел офисных компьютеров", result)
        self.assertIn("графический отдел", result)

```

```

expected_gaming_computers = ["ASUS ROG Strix G15", "MSI Gaming GE76"]
self.assertEqual(result["отдел игровых компьютеров"], expected_gaming_computers)

def test_query_2(self):
    """Тест: Средняя цена по дисплейным классам"""
    result = query_2(self.display_classes, self.one_to_many)
    self.assertIsInstance(result, list)
    self.assertEqual(len(result), 4)

    prices = [avg_price for _, avg_price in result]
    self.assertEqual(prices, sorted(prices))

    for name, avg_price in result:
        if name == "графический отдел":
            expected_avg = (210000 + 195000) / 2
            self.assertAlmostEqual(avg_price, expected_avg, places=2)

def test_query_3(self):
    """Тест: Компьютеры, начинающиеся на 'A'"""
    result = query_3(self.many_to_many)
    self.assertIsInstance(result, dict)

    self.assertEqual(len(result), 3)

    self.assertIn("ASUS ROG Strix G15", result)
    self.assertIn("Acer Aspire 5", result)
    self.assertIn("Asus VivoBook", result)

    self.assertNotIn("Dell OptiPlex 3090", result)
    self.assertNotIn("HP EliteBook 840", result)

    self.assertEqual(result["ASUS ROG Strix G15"], ["отдел игровых компьютеров"])

    self.assertEqual(result["Acer Aspire 5"], ["учебный класс"])

def main():
    display_classes, computers, comp_display_classes = create_data()
    one_to_many = get_one_to_many(display_classes, computers)
    many_to_many = get_many_to_many(display_classes, computers, comp_display_classes)

    print("Запрос 1. Дисплейные классы со словом 'отдел' в названии")
    result1 = query_1(display_classes, one_to_many)
    for dc_name, computer_list in result1.items():

```

```

print(f'{dc_name}: {' '.join(computer_list)}")

print("\nЗапрос 2. Дисплейные классы по средней цене компьютеров (от меньшей к
большой)")
result2 = query_2(display_classes, one_to_many)
for dc_name, avg_price in result2:
    print(f'{dc_name}: {avg_price}")

print("\nЗапрос 3: Компьютеры, модель которых начинается на 'A' (латинская)")
result3 = query_3(many_to_many)
for computer_model, dc_list in result3.items():
    print(f"Компьютер '{computer_model}' находится в дисплейных классах: {' '.join(dc_list)}")

if __name__ == '__main__':
    main()

```

Примеры выполнения программы

```

/Users/ruslangadziev/PycharmProjects/IU5/.venv/bin/python /Applications/PyCharm CE.app/Contents/plugins/python-ce/
Testing started at 05:35 ...
Launching unittests with arguments python -m unittest /Users/ruslangadziev/PycharmProjects/IU5/RK2/main.py in /Use

Ran 3 tests in 0.003s

OK

Process finished with exit code 0

```