Московский государственный технический
университет им. Н. Э. Баумана
Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования» Отчет
по рубежному контролю №2
«**Вариант А, 7**»

Выполнил:
Студент группы ИУ5-31Б
Клубков Максим

Проверил:
Гапанюк Ю. Е.

2025 г.

Листинг программы

Rk2_varA7.py

```
from operator import itemgetter
from typing import List, Dict, Tuple


class Microprocessor:
    def __init__(self, id: int, model: str, frequency: int, computer_id:
int):
        self.id = id
        self.model = model
        self.frequency = frequency
        self.computer_id = computer_id


class Computer:
    def __init__(self, id: int, name: str):
        self.id = id
        self.name = name


class MicroprocessorComputer:
    def __init__(self, computer_id: int, microprocessor_id: int):
        self.computer_id = computer_id
        self.microprocessor_id = microprocessor_id


computers = [
    Computer(1, 'отдел кадров'),
    Computer(2, 'архивный отдел'),
    Computer(3, 'бухгалтерия'),
    Computer(11, 'отдел разработки'),
    Computer(22, 'архивный отдел тестирования'),
    Computer(33, 'бухгалтерия финансов'),
]

microprocessors = [
    Microprocessor(1, 'Intel Core i7', 3700, 1),
    Microprocessor(2, 'Intel Core i5', 3200, 2),
    Microprocessor(3, 'Intel Xeon', 4200, 3),
    Microprocessor(4, 'AMD Ryzen 7', 3800, 3),
    Microprocessor(5, 'AMD Ryzen 5', 3400, 3),
]

microprocessors_computers = [
    MicroprocessorComputer(1, 1),
    MicroprocessorComputer(2, 2),
    MicroprocessorComputer(3, 3),
    MicroprocessorComputer(3, 4),
    MicroprocessorComputer(3, 5),
    MicroprocessorComputer(11, 1),
    MicroprocessorComputer(22, 2),
    MicroprocessorComputer(33, 3),
    MicroprocessorComputer(33, 4),
    MicroprocessorComputer(33, 5),
]
```

```python
class DataService:

    @staticmethod
    def create_one_to_many_relation(
            computers_list: List[Computer],
            microprocessors_list: List[Microprocessor]
    ) -> List[Tuple[str, int, str]]:
        return [
            (m.model, m.frequency, c.name)
            for c in computers_list
            for m in microprocessors_list
            if m.computer_id == c.id
        ]

    @staticmethod
    def create_many_to_many_relation(
            computers_list: List[Computer],
            microprocessors_list: List[Microprocessor],
            relations_list: List[MicroprocessorComputer]
    ) -> List[Tuple[str, int, str]]:
        many_to_many_temp = [
            (c.name, mc.computer_id, mc.microprocessor_id)
            for c in computers_list
            for mc in relations_list
            if c.id == mc.computer_id
        ]

        return [
            (m.model, m.frequency, comp_name)
            for comp_name, comp_id, micro_id in many_to_many_temp
            for m in microprocessors_list if m.id == micro_id
        ]

    @staticmethod
    def task_a1_one_to_many_sorted(
            one_to_many_data: List[Tuple[str, int, str]]
    ) -> List[Tuple[str, int, str]]:
        return sorted(one_to_many_data, key=itemgetter(2))

    @staticmethod
    def task_a2_computers_total_frequency(
            one_to_many_data: List[Tuple[str, int, str]],
            computers_list: List[Computer]
    ) -> List[Tuple[str, int]]:
        result_unsorted = []

        computer_groups = {}
        for m_model, m_freq, c_name in one_to_many_data:
            if c_name not in computer_groups:
                computer_groups[c_name] = []
            computer_groups[c_name].append(m_freq)

        for c_name, frequencies in computer_groups.items():
            total_frequency = sum(frequencies)
            result_unsorted.append((c_name, total_frequency))

        return sorted(result_unsorted, key=itemgetter(1), reverse=True)

    @staticmethod
    def task_a3_computers_with_department_and_microprocessors(
            computers_list: List[Computer],
            many_to_many_data: List[Tuple[str, int, str]]
    ) -> Dict[str, List[str]]:
```

```
        result = {}

        for computer in computers_list:
            if 'отдел' in computer.name.lower():
                computer_microprocessors = [
                    m_model
                    for m_model, m_freq, comp_name in many_to_many_data
                    if comp_name == computer.name
                ]
                result[computer.name] = computer_microprocessors

        return result


def main():
    service = DataService()

    one_to_many = service.create_one_to_many_relation(computers,
microprocessors)
    many_to_many = service.create_many_to_many_relation(
        computers, microprocessors, microprocessors_computers
    )

    print('Задание А1')
    print('Список всех связанных микропроцессоров и компьютеров,
отсортированный по компьютерам:')
    task1_result = service.task_a1_one_to_many_sorted(one_to_many)
    for item in task1_result:
        print(f'Микропроцессор: {item[0]}, Частота: {item[1]}, Компьютер:
{item[2]}')

    print('\n' + '=' * 50 + '\n')

    print('Задание А2')
    print('Список компьютеров с суммарной частотой микропроцессоров,
отсортированный по суммарной частоте:')
    task2_result = service.task_a2_computers_total_frequency(one_to_many,
computers)
    for item in task2_result:
        print(f'Компьютер: {item[0]}, Суммарная частота: {item[1]}')

    print('\n' + '=' * 50 + '\n')

    print('Задание А3')
    print('Список всех компьютеров с "отдел" в названии и их
микропроцессоры:')
    task3_result =
service.task_a3_computers_with_department_and_microprocessors(
        computers, many_to_many
    )
    for computer_name, microprocessors_list in task3_result.items():
        print(f'Компьютер: {computer_name}')
        print(f'  Микропроцессоры: {", ".join(microprocessors_list)}')


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

Test_rk2_varA7.py

```python
import unittest
from rk2_varA7 import (
    Computer,
    Microprocessor,
    MicroprocessorComputer,
    DataService
)


class TestDataService(unittest.TestCase):

    def setUp(self):
        self.computers = [
            Computer(1, 'отдел кадров'),
            Computer(2, 'архивный отдел'),
            Computer(3, 'бухгалтерия'),
        ]

        self.microprocessors = [
            Microprocessor(1, 'Intel Core i7', 3700, 1),
            Microprocessor(2, 'Intel Core i5', 3200, 2),
            Microprocessor(3, 'Intel Xeon', 4200, 3),
        ]

        self.relations = [
            MicroprocessorComputer(1, 1),
            MicroprocessorComputer(2, 2),
            MicroprocessorComputer(3, 3),
        ]

        self.service = DataService()

    def test_create_one_to_many_relation(self):
        result = self.service.create_one_to_many_relation(
            self.computers, self.microprocessors
        )

        self.assertEqual(len(result), 3)

        expected_data = [
            ('Intel Core i7', 3700, 'отдел кадров'),
            ('Intel Core i5', 3200, 'архивный отдел'),
            ('Intel Xeon', 4200, 'бухгалтерия'),
        ]

        for expected, actual in zip(expected_data, result):
            self.assertEqual(expected[0], actual[0])
            self.assertEqual(expected[1], actual[1])
            self.assertEqual(expected[2], actual[2])

    def test_task_a1_one_to_many_sorted(self):
        test_data = [
            ('AMD Ryzen', 3500, 'бухгалтерия'),
            ('Intel Core i7', 3700, 'отдел кадров'),
            ('Intel Core i5', 3200, 'архивный отдел'),
        ]

        result = self.service.task_a1_one_to_many_sorted(test_data)

        expected_order = ['архивный отдел', 'бухгалтерия', 'отдел кадров']
        actual_order = [item[2] for item in result]
```

```
        self.assertEqual(expected_order, actual_order)
        self.assertEqual(len(result), 3)

    def test_task_a2_computers_total_frequency(self):
        test_data = [
            ('Intel Core i7', 3700, 'отдел кадров'),
            ('Intel Core i5', 3200, 'архивный отдел'),
            ('Intel Xeon', 4200, 'бухгалтерия'),
            ('AMD Ryzen 7', 3800, 'бухгалтерия'),
        ]

        test_computers = [
            Computer(1, 'отдел кадров'),
            Computer(2, 'архивный отдел'),
            Computer(3, 'бухгалтерия'),
        ]

        result = self.service.task_a2_computers_total_frequency(
            test_data, test_computers
        )

        self.assertEqual(len(result), 3)

        expected_results = {
            'бухгалтерия': 8000,
            'отдел кадров': 3700,
            'архивный отдел': 3200,
        }

        for computer_name, total_freq in result:
            self.assertEqual(expected_results[computer_name], total_freq)

        frequencies = [freq for _, freq in result]
        self.assertEqual(frequencies, sorted(frequencies, reverse=True))

    def test_task_a3_computers_with_department_and_microprocessors(self):
        test_computers = [
            Computer(1, 'отдел кадров'),
            Computer(2, 'архивный отдел'),
            Computer(3, 'бухгалтерия'),
            Computer(4, 'отдел разработки'),
        ]

        test_many_to_many = [
            ('Intel Core i7', 3700, 'отдел кадров'),
            ('Intel Core i5', 3200, 'архивный отдел'),
            ('Intel Xeon', 4200, 'бухгалтерия'),
            ('AMD Ryzen 7', 3800, 'отдел кадров'),
            ('AMD Ryzen 5', 3400, 'отдел разработки'),
        ]

        result =
self.service.task_a3_computers_with_department_and_microprocessors(
            test_computers, test_many_to_many
        )

        self.assertEqual(len(result), 3)
        self.assertIn('отдел кадров', result)
        self.assertIn('архивный отдел', result)
        self.assertIn('отдел разработки', result)
        self.assertNotIn('бухгалтерия', result)
```

```
        expected_microprocessors = ['Intel Core i7', 'AMD Ryzen 7']
        self.assertEqual(sorted(result['отдел кадров']),
sorted(expected_microprocessors))

    def test_create_many_to_many_relation(self):
        result = self.service.create_many_to_many_relation(
            self.computers, self.microprocessors, self.relations
        )

        self.assertEqual(len(result), 3)

        for item in result:
            self.assertIsInstance(item, tuple)
            self.assertEqual(len(item), 3)
            self.assertIsInstance(item[0], str)
            self.assertIsInstance(item[1], int)
            self.assertIsInstance(item[2], str)


if __name__ == '__main__':
    unittest.main(verbosity=2)
```

Результат выполнения

```
D:\analytics\.venv\пикап\Scripts\python.exe D:\пикап\rk2_varA7.py
Задание A1
Список всех связанных микропроцессоров и компьютеров, отсортированный по компьютерам:
Микропроцессор: Intel Core i5, Частота: 3200, Компьютер: архивный отдел
Микропроцессор: Intel Xeon, Частота: 4200, Компьютер: бухгалтерия
Микропроцессор: AMD Ryzen 7, Частота: 3800, Компьютер: бухгалтерия
Микропроцессор: AMD Ryzen 5, Частота: 3400, Компьютер: бухгалтерия
Микропроцессор: Intel Core i7, Частота: 3700, Компьютер: отдел кадров

==================================================

Задание A2
Список компьютеров с суммарной частотой микропроцессоров, отсортированный по суммарной частоте:
Компьютер: бухгалтерия, Суммарная частота: 11400
Компьютер: отдел кадров, Суммарная частота: 3700
Компьютер: архивный отдел, Суммарная частота: 3200

==================================================

Задание A3
Список всех компьютеров с "отдел" в названии и их микропроцессоры:
Компьютер: отдел кадров
  Микропроцессоры: Intel Core i7
Компьютер: архивный отдел
  Микропроцессоры: Intel Core i5
Компьютер: отдел разработки
  Микропроцессоры: Intel Core i7
Компьютер: архивный отдел тестирования
  Микропроцессоры: Intel Core i5

Process finished with exit code 0
```


Tests passed: 5 of 5 tests – 33 ms
D:\analytics\.venv\пикап\Scripts\python.exe "C:/Program Files/JetBrains/PyCharm Community Edition 2024.1.4/plugins/python-ce/helpers/pycha
Testing started at 11:43 ...
Launching unittests with arguments python -m unittest D:\пикап\test_rk2_varA7.py in D:\пикап
Ran 5 tests in 0.035s
OK
Process finished with exit code 0