main_refactored:

```
class Computer:
    def __init__(self, id: int, name: str, cost: int, display_class_id: int = None):
        self.id = id
        self.name = name
        self.threads = cost
        self.display_class_id = display_class_id


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


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


def list_computers_with_name_ending_ov(computers, display_classes, links):
    result = []
    for link in links:
        computer = next(c for c in computers if c.id == link.computer_id)
        if computer.name.endswith("n"):
            display_class = next(d for d in display_classes if d.id ==
link.display_class_id)
            result.append((computer.name, display_class.name))
    return result


def list_average_threads_per_display(display_classes, computers, links):
    avg_threads_per_class = []
    for display_class in display_classes:
        related_computers = [
            c for c in computers if any(l.display_class_id == display_class.id and
l.computer_id == c.id for l in links)
        ]
        if related_computers:
```

```
            total_threads = sum(c.threads for c in related_computers)
            avg_threads = total_threads / len(related_computers)
            avg_threads_per_class.append((display_class.name, avg_threads))
    avg_threads_per_class.sort(key=lambda x: x[1])
    return avg_threads_per_class


def list_display_classes_starting_with_n(display_classes, computers, links):
    result = []
    for display_class in display_classes:
        if display_class.name.startswith("N"):
            related_computers = [
                c for c in computers if any(l.display_class_id == display_class.id
and l.computer_id == c.id for l in links)
            ]
            computer_names = [c.name for c in related_computers]
            result.append((display_class.name, computer_names))
    return result
```

test_main:

```
import unittest
from main_refactored import Computer, DisplayClass,
ComputerAndDisplayClass,list_computers_with_name_ending_ov,
list_average_threads_per_display, list_display_classes_starting_with_n


class TestMain(unittest.TestCase):


    def setUp(self):
        self.display_classes = [
            DisplayClass(id=1, name="Quantum"),
            DisplayClass(id=2, name="Cyber"),
            DisplayClass(id=3, name="Nano"),
            DisplayClass(id=4, name="Titan"),
            DisplayClass(id=5, name="Nexus")
        ]


        self.computers = [
```

```
            Computer(id=1, name="Alpha", cost=50000),
            Computer(id=2, name="Beta", cost=20000),
            Computer(id=3, name="Gamma", cost=70000),
            Computer(id=4, name="Delta", cost=90000),
            Computer(id=5, name="Epsilon", cost=30000)
        ]

        self.links = [
            ComputerAndDisplayClass(display_class_id=1, computer_id=1),
            ComputerAndDisplayClass(display_class_id=1, computer_id=3),
            ComputerAndDisplayClass(display_class_id=2, computer_id=2),
            ComputerAndDisplayClass(display_class_id=3, computer_id=3),
            ComputerAndDisplayClass(display_class_id=4, computer_id=4),
            ComputerAndDisplayClass(display_class_id=5, computer_id=1),
            ComputerAndDisplayClass(display_class_id=5, computer_id=5),
            ComputerAndDisplayClass(display_class_id=2, computer_id=4)
        ]

    def test_list_computers_with_name_ending_ov(self):
        result = list_computers_with_name_ending_ov(self.computers,
self.display_classes, self.links)
        expected = [('Epsilon', 'Nexus')]
        self.assertEqual(result, expected)

    def test_list_average_threads_per_display(self):
        result = list_average_threads_per_display(self.display_classes,
self.computers, self.links)
        expected = [('Nexus', 40000.0), ('Cyber', 55000.0), ('Quantum', 60000.0),
('Nano', 70000.0), ('Titan', 90000.0)]
        self.assertEqual(result, expected)

    def test_list_display_classes_starting_with_n(self):
        result = list_display_classes_starting_with_n(self.display_classes,
self.computers, self.links)
        expected = [
            ('Nano', ['Gamma']),
            ('Nexus', ['Alpha', 'Epsilon'])
```

```
        ]
        self.assertEqual(result, expected)

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