Отрефакторированный код

```
class HardDrive:
    def __init__(self, id: int, model: str, capacity: int, computer_id: int):
        self.id = id
        self.model = model
        self.capacity = capacity
        self.computer_id = computer_id


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


class ComputerHardDrive:
    def __init__(self, computer_id: int, hard_drive_id: int):
        self.computer_id = computer_id
        self.hard_drive_id = hard_drive_id


def create_one_to_many(computers, hard_drives):
    return [
        (hd.model, hd.capacity, comp.name)
        for comp in computers
        for hd in hard_drives
        if hd.computer_id == comp.id
    ]


def create_many_to_many(computers, computer_hard_drives, hard_drives):
    many_to_many_temp = [
```

```
        (comp.name, elem.computer_id, elem.hard_drive_id)
        for comp in computers
        for elem in computer_hard_drives
        if comp.id == elem.computer_id
    ]
    return [
        (hd.model, hd.capacity, comp_name)
        for comp_name, computer_id, hard_drive_id in many_to_many_temp
        for hd in hard_drives
        if hd.id == hard_drive_id
    ]

def task1(one_to_many):
    filtered = filter(lambda x: x[2].startswith('A'), one_to_many)
    return [(model, comp) for model, _, comp in filtered]

def task2(computers, one_to_many):
    res2 = []
    for comp in computers:
        drives = filter(lambda x: x[2] == comp.name, one_to_many)
        drives = list(drives)
        if drives:
            max_capacity = max([capacity for _, capacity, _ in drives])
            res2.append((comp.name, max_capacity))
    return sorted(res2, key=lambda x: x[1], reverse=True)

def task3(many_to_many):
    sorted_data = sorted(many_to_many, key=lambda x: x[2])
    return [(model, comp) for model, _, comp in sorted_data]
```

```
if __name__ == '__main__':
    # Исходные данные
    computers = [
        Computer(1, 'AlphaPC'),
        Computer(2, 'BetaPC'),
        Computer(3, 'GammaPC')
    ]

    hard_drives = [
        HardDrive(1, 'Seagate', 1000, 1),
        HardDrive(2, 'WD Blue', 2000, 1),
        HardDrive(3, 'Samsung EVO', 500, 2),
        HardDrive(4, 'Toshiba', 750, 3),
        HardDrive(5, 'Hitachi', 1500, 3)
    ]

    computer_hard_drives = [
        ComputerHardDrive(1, 1),
        ComputerHardDrive(1, 2),
        ComputerHardDrive(2, 3),
        ComputerHardDrive(3, 4),
        ComputerHardDrive(3, 5)
    ]

    one_to_many = create_one_to_many(computers, hard_drives)
    many_to_many = create_many_to_many(computers, computer_hard_drives, hard_drives)

    print("Task 1:", task1(one_to_many))
    print("Task 2:", task2(computers, one_to_many))
    print("Task 3:", task3(many_to_many))
```

Модульные тесты (TDD)

файл test_rub_control.py с тестами.

```
import unittest
from main import (
    create_one_to_many,
    create_many_to_many,
    task1,
    task2,
    task3,
    Computer,
    HardDrive,
    ComputerHardDrive
)


class TestRubControl(unittest.TestCase):
    def setUp(self):
        self.computers = [
            Computer(1, 'AlphaPC'),
            Computer(2, 'BetaPC'),
            Computer(3, 'GammaPC')
        ]
        self.hard_drives = [
            HardDrive(1, 'Seagate', 1000, 1),
            HardDrive(2, 'WD Blue', 2000, 1),
            HardDrive(3, 'Samsung EVO', 500, 2),
            HardDrive(4, 'Toshiba', 750, 3),
```

```
            HardDrive(5, 'Hitachi', 1500, 3)
        ]
        self.computer_hard_drives = [
            ComputerHardDrive(1, 1),
            ComputerHardDrive(1, 2),
            ComputerHardDrive(2, 3),
            ComputerHardDrive(3, 4),
            ComputerHardDrive(3, 5)
        ]

    def test_create_one_to_many(self):
        result = create_one_to_many(self.computers, self.hard_drives)
        expected = [
            ('Seagate', 1000, 'AlphaPC'),
            ('WD Blue', 2000, 'AlphaPC'),
            ('Samsung EVO', 500, 'BetaPC'),
            ('Toshiba', 750, 'GammaPC'),
            ('Hitachi', 1500, 'GammaPC')
        ]
        self.assertEqual(result, expected)

    def test_create_many_to_many(self):
        result = create_many_to_many(
            self.computers, self.computer_hard_drives, self.hard_drives
        )
        expected = [
            ('Seagate', 1000, 'AlphaPC'),
            ('WD Blue', 2000, 'AlphaPC'),
            ('Samsung EVO', 500, 'BetaPC'),
            ('Toshiba', 750, 'GammaPC'),
```

```
        ('Hitachi', 1500, 'GammaPC')
    ]
    self.assertEqual(result, expected)

def test_task1(self):
    one_to_many = create_one_to_many(self.computers, self.hard_drives)
    result = task1(one_to_many)
    expected = [('Seagate', 'AlphaPC'), ('WD Blue', 'AlphaPC')]
    self.assertEqual(result, expected)

def test_task2(self):
    one_to_many = create_one_to_many(self.computers, self.hard_drives)
    result = task2(self.computers, one_to_many)
    expected = [('AlphaPC', 2000), ('GammaPC', 1500), ('BetaPC', 500)]
    self.assertEqual(result, expected)

def test_task3(self):
    many_to_many = create_many_to_many(
        self.computers, self.computer_hard_drives, self.hard_drives
    )
    result = task3(many_to_many)
    expected = [
        ('Seagate', 'AlphaPC'),
        ('WD Blue', 'AlphaPC'),
        ('Samsung EVO', 'BetaPC'),
        ('Toshiba', 'GammaPC'),
        ('Hitachi', 'GammaPC')
    ]
    self.assertEqual(result, expected)
```

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

Пример вывода тестов

```
.....
----------------------------------------
Ran 5 tests in 0.001s

OK
```