**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
«Модульное тестирование»

Выполнил:
Студент группы ИУ5-31Б
Куртинец Роман

Проверил:
Гапанюк Ю. Е.

2025 г.

Листинг программы

main.py

```python
# main.py
from typing import List, Any, Callable, Dict
from dataclasses import dataclass
from collections import defaultdict

@dataclass
class HardDrive:
    hdd_id: int
    model: str
    capacity: int
    computer_id: int

@dataclass
class Computer:
    comp_id: int
    name: str

@dataclass
class HardDriveComputer:
    hard_drive_id: int
    computer_id: int

def get_computers() -> List[Computer]:
    """ Функция для получения данных компьютеров. """
    return [
        Computer(1, "Компьютер Dell XPS"),
        Computer(2, "Компьютер HP Pavilion"),
        Computer(3, "Сервер Lenovo"),
        Computer(4, "Компьютер Apple MacBook")
    ]

def get_hard_drives() -> List[HardDrive]:
    """ Функция для получения данных жёстких дисков. """
    return [
        HardDrive(1, "Seagate 1TB", 1000, 1),
        HardDrive(2, "WD 500GB", 500, 1),
        HardDrive(3, "Samsung 2TB", 2000, 2),
        HardDrive(4, "Crucial 1TB SSD", 1000, 3),
        HardDrive(5, "Toshiba 750GB", 750, 2)
    ]
```

```python

def get_hard_drive_computers() -> List[HardDriveComputer]:
    """ Функция для получения связей многие ко многим. """
    return [
        HardDriveComputer(1, 1),
        HardDriveComputer(1, 4),
        HardDriveComputer(2, 1),
        HardDriveComputer(3, 2),
        HardDriveComputer(4, 3),
        HardDriveComputer(5, 2),
        HardDriveComputer(5, 4)
    ]

def print_data(data: List[Any], headers: List[str], title: str, column_width: int = 30) -> None:
    """
    Функция для вывода данных в виде таблицы.
    Принимает данные, заголовки столбцов и заголовок таблицы.
    """
    total_length = len(headers) * column_width
    columns = len(headers)

    print(f"{title:=^{total_length}}")
    print(("{:<{column_width}}" * columns).format(*headers, column_width=column_width))
    print()
    print("\n".join(
        ("{:<{column_width}}" * columns).format(*i, column_width=column_width) for i in data
    ))
    print()

def first_query(computers: List[Computer], hard_drives: List[HardDrive]) -> List[Any]:
    """ Реализация первого запроса: Связанные объекты (один-ко-многим). """
    computer_dict = {c.comp_id: c for c in computers}

    joined = [(computer_dict[hd.computer_id].name, hd.model)
              for hd in hard_drives
              if hd.computer_id in computer_dict]

    return sorted(joined, key=lambda x: x[0])

def second_query(computers: List[Computer], hard_drives: List[HardDrive]) -> List[Any]:
```

```python
    """ Реализация второго запроса: Суммарная ёмкость (один-ко-многим). """
    computer_dict = {c.comp_id: c for c in computers}

    capacity_sum = defaultdict(int)
    for hd in hard_drives:
        capacity_sum[hd.computer_id] += hd.capacity

    comp_capacities = [(computer_dict[cid].name, total)
                       for cid, total in capacity_sum.items()
                       if cid in computer_dict]

    return sorted(comp_capacities, key=lambda x: x[1], reverse=True)

def third_query(computers: List[Computer], hard_drives: List[HardDrive],
                relations: List[HardDriveComputer], condition: Callable) -> List[Any]:
    """ Реализация третьего запроса: Фильтрация по условию (многие-ко-многим). """
    filtered_computers = [c for c in computers
                          if condition(c.name.lower())]

    hard_drive_dict = {hd.hdd_id: hd for hd in hard_drives}

    comp_hds = defaultdict(list)
    for link in relations:
        hd = hard_drive_dict.get(link.hard_drive_id)
        if hd:
            comp_hds[link.computer_id].append(hd)

    result = []
    for comp in filtered_computers:
        for hd in comp_hds.get(comp.comp_id, []):
            result.append((comp.name, hd.model, hd.capacity))

    return result

def main() -> None:
    computers = get_computers()
    hard_drives = get_hard_drives()
    relations = get_hard_drive_computers()

    # Первый запрос
    print_data(first_query(computers, hard_drives),
               ["Компьютер", "Жёсткий диск"], "Запрос 1")

    # Второй запрос
    print_data(second_query(computers, hard_drives),
               ["Компьютер", "Суммарная ёмкость (ГБ)"], "Запрос 2")

    # Третий запрос
    print_data(third_query(computers, hard_drives, relations,
                           lambda name: "компьютер" in name),
               ["Компьютер", "Жёсткий диск", "Ёмкость (ГБ)"], "Запрос 3")

if __name__ == "__main__":
    main()

```

test_main.py

```python
# test_main.py
import pytest
from pytest_unordered import unordered

from main import (
    HardDrive, Computer, HardDriveComputer,
    first_query, second_query, third_query
)

@pytest.fixture
def test_one_to_many_data():
    computers = [
        Computer(1, "Компьютер A"),
        Computer(2, "Компьютер B"),
        Computer(3, "Сервер C"),
    ]
    hard_drives = [
        HardDrive(1, "HDD1", 1000, 1),
        HardDrive(2, "HDD2", 500, 1),
        HardDrive(3, "HDD3", 2000, 2),
        HardDrive(4, "HDD4", 750, 3),
        HardDrive(5, "HDD5", 1000, 2),
    ]
    return computers, hard_drives

@pytest.fixture
def test_many_to_many_data():
    computers = [
        Computer(1, "Компьютер A"),
        Computer(2, "Компьютер B"),
        Computer(3, "Сервер C"),
        Computer(4, "Компьютер D"),
    ]
    hard_drives = [
        HardDrive(1, "HDD1", 1000, -1),
        HardDrive(2, "HDD2", 500, -1),
        HardDrive(3, "HDD3", 2000, -1),
        HardDrive(4, "HDD4", 750, -1),
        HardDrive(5, "HDD5", 1000, -1),
    ]
```

```python
    relations = [
        HardDriveComputer(1, 1),
        HardDriveComputer(1, 2),
        HardDriveComputer(2, 1),
        HardDriveComputer(3, 2),
        HardDriveComputer(4, 3),
        HardDriveComputer(5, 2),
        HardDriveComputer(5, 4),
        HardDriveComputer(3, 4),
        HardDriveComputer(2, 3),
    ]
    return computers, hard_drives, relations

def test_first_query(test_one_to_many_data):
    computers, hard_drives = test_one_to_many_data
    expected = [
        ("Компьютер A", "HDD1"),
        ("Компьютер A", "HDD2"),
        ("Компьютер B", "HDD3"),
        ("Компьютер B", "HDD5"),
        ("Сервер C", "HDD4"),
    ]
    result = first_query(computers, hard_drives)
    assert result == expected

def test_second_query(test_one_to_many_data):
    computers, hard_drives = test_one_to_many_data
    expected = [
        ("Компьютер B", 3000),
        ("Компьютер A", 1500),
        ("Сервер C", 750),
    ]
    result = second_query(computers, hard_drives)
    assert result == expected

def test_third_query(test_many_to_many_data):
    computers, hard_drives, relations = test_many_to_many_data
```

```python
    expected = unordered([
        ("Компьютер A", "HDD1", 1000),
        ("Компьютер A", "HDD2", 500),
        ("Компьютер B", "HDD1", 1000),
        ("Компьютер B", "HDD3", 2000),
        ("Компьютер B", "HDD5", 1000),
        ("Компьютер D", "HDD3", 2000),
        ("Компьютер D", "HDD5", 1000),
    ])
    result = third_query(computers, hard_drives, relations,
                         lambda name: "компьютер" in name)
    assert result == expected
```

Результат выполнения

```
(venv) at0r@Roma-M14:~/Kurtinets_Labs_3sem$ pytest -v
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-9.0.2, pluggy-1.6.0 -- /home/at0r/Kurtinets_Labs_3sem/venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/at0r/Kurtinets_Labs_3sem
plugins: unordered-0.7.0
collected 3 items

test_main.py::test_first_query PASSED
test_main.py::test_second_query PASSED
test_main.py::test_third_query PASSED

============================== 3 passed in 0.03s ===============================
```