Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

Факультет «Информатика и системы управления»

Курс «Базовые компоненты интернет-технологий»

Рубежный контроль №2

Выполнил:

студент группы ИУ5-33Б Николай Горкунов

подпись: , дата:

Проверил:

преподаватель Юрий Гапанюк

подпись: , дата:

2022 г.

Задание.

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).

Рубежный контроль №1 представлял собой разработку программы на языке Python, которая выполняет следующие действия:

1) Необходимо создать два класса данных в соответствии с Вашим вариантом предметной области, которые связаны отношениями один-ко-многим и многие-ко-многим.
   Пример классов данных для предметной области Сотрудник-Отдел:
   1. Класс «Сотрудник», содержащий поля:
      - ID записи о сотруднике;
      - Фамилия сотрудника;
      - Зарплата (количественный признак);
      - ID записи об отделе. (для реализации связи один-ко-многим)
   2. Класс «Отдел», содержащий поля:
      - ID записи об отделе;
      - Наименование отдела.
   3. (Для реализации связи многие-ко-многим) Класс «Сотрудники отдела», содержащий поля:
      - ID записи о сотруднике;
      - ID записи об отделе.
2) Необходимо создать списки объектов классов, содержащих тестовые данные (3-5 записей), таким образом, чтобы первичные и вторичные ключи соответствующих записей были связаны по идентификаторам.
3) Необходимо разработать запросы в соответствии с Вашим вариантом. Запросы сформулированы в терминах классов «Сотрудник» и «Отдел», которые используются в примере. Вам нужно перенести эти требования в Ваш вариант предметной области. При разработке запросов необходимо по возможности использовать функциональные возможности языка Python (list/dict comprehensions, функции высших порядков). Для реализации запроса №2 введите в класс, находящийся на стороне связи «много», произвольный количественный признак, например, «зарплата сотрудника».

Для моей группы вариант запросов В, для меня вариант предметной области 4.

1) Вариант В запросов:
   1. «Отдел» и «Сотрудник» связаны отношением один-ко-многим. Выведите список всех сотрудников, у которых фамилия начинается с буквы «А», и названия их отделов.
   2. «Отдел» и «Сотрудник» связаны отношением один-ко-многим. Выведите список отделов с минимальной зарплатой сотрудников в каждом отделе, отсортированный по минимальной зарплате.
   3. «Отдел» и «Сотрудник» связаны отношением многие-ко-многим. Выведите список всех связанных сотрудников и отделов, отсортированный по сотрудникам, сортировка по отделам произвольная.
2) Вариант 4 предметной области:
   Компьютер – Дисплейный класс.

Исходный код.

modules/DataBase.py:

```
from operator import itemgetter

class Computer:
    """Компьютер"""
    def __init__(self, id, name, cost, clas_id):
        self.id = id
        self.name = name
        self.cost = cost
        self.clas_id = clas_id

class DispClass:
    """Дисплейный класс"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class ComputerDispClass:
    """
    'Компьютеры дисплейного класса' для реализации
    связи многие-ко-многим
    """
    def __init__(self, clas_id, comp_id):
        self.clas_id = clas_id
        self.comp_id = comp_id

# Дисплейный классы
classes = [
    DispClass(1, 'Класс информатики'),
    DispClass(2, 'Класс генной инженерии'),
    DispClass(3, 'Класс 3Д моделирования'),

    DispClass(11, 'Класс информатики (другой)'),
    DispClass(22, 'Класс генной инженерии (другой)'),
    DispClass(33, 'Класс 3Д моделирования (другой)'),
]

# Компьютеры
computers = [
    Computer(1, 'Делл_АУМ01', 125000, 1),
    Computer(2, 'Эпл__АУМ02', 135000, 2),
```

```
    Computer(3, 'Асус_АУМ03', 145000, 3),
    Computer(4, 'Делл_АУМ04', 135000, 3),
    Computer(5, 'Асус_АУМ05', 125000, 3),
]

# Данные о связи многие-ко-многим
computers_classes = [
    ComputerDispClass(1, 1),
    ComputerDispClass(2, 2),
    ComputerDispClass(3, 3),
    ComputerDispClass(3, 4),
    ComputerDispClass(3, 5),

    ComputerDispClass(11, 1),
    ComputerDispClass(22, 2),
    ComputerDispClass(33, 3),
    ComputerDispClass(33, 4),
    ComputerDispClass(33, 5),
]

class DataBase:
    """
    БД с исходными и промежуточными данными
    """
    def __init__(self, classes, computers, computers_classes):
        self.classes = classes
        self.computers = computers
        self.computers_classes = computers_classes
        self.one_to_many = [(comp.name, comp.cost, clas.name)
            for clas in classes
            for comp in computers
            if comp.clas_id == clas.id]
        many_to_many_tmp = [(clas.name, compclas.clas_id,
compclas.comp_id)
            for clas in classes
            for compclas in computers_classes
            if clas.id == compclas.clas_id]
        self.many_to_many = [(comp.name, comp.cost, clas_name)
            for clas_name, clas_id, comp_id in many_to_many_tmp
            for comp in computers if comp.id == comp_id]

    def task_b1(self):
```

```
        res_11 = list(filter(lambda x: x[0].find('A') == 0,
self.one_to_many))
        return res_11

    def task_b2(self):
        res_12_unsorted = []
        for clas in classes:
            clas_comps = list(filter(lambda i: i[2] == clas.name,
self.one_to_many))
            if len(clas_comps) > 0:
                clas_costs = list(map(lambda x: x[1], clas_comps))
                clas_costs_min = min(clas_costs)
                res_12_unsorted.append((clas.name,
clas_costs_min))
        res_12 = sorted(res_12_unsorted, key = itemgetter(1))
        return res_12

    def task_b3(self):
        return sorted(self.many_to_many, key = itemgetter(0))
```

test_tasks.py:

```
import pytest
from modules.DataBase import *

# Дисплейный классы
classes = [
    DispClass(1, 'Класс информатики'),
    DispClass(2, 'Класс генной инженерии'),
    DispClass(3, 'Класс 3Д моделирования'),

    DispClass(11, 'Класс информатики (другой)'),
    DispClass(22, 'Класс генной инженерии (другой)'),
    DispClass(33, 'Класс 3Д моделирования (другой)'),
]

# Компьютеры
computers = [
    Computer(1, 'Делл_АУМ01', 125000, 1),
    Computer(2, 'Эпл__АУМ02', 135000, 2),
    Computer(3, 'Асус_АУМ03', 145000, 3),
    Computer(4, 'Делл_АУМ04', 135000, 3),
    Computer(5, 'Асус_АУМ05', 125000, 3),
```

```
]

# Данные о связи многие-ко-многим
computers_classes = [
    ComputerDispClass(1, 1),
    ComputerDispClass(2, 2),
    ComputerDispClass(3, 3),
    ComputerDispClass(3, 4),
    ComputerDispClass(3, 5),

    ComputerDispClass(11, 1),
    ComputerDispClass(22, 2),
    ComputerDispClass(33, 3),
    ComputerDispClass(33, 4),
    ComputerDispClass(33, 5),
]

def test_task_b1():
    db = DataBase(classes, computers, computers_classes)
    assert db.task_b1() == [
        ('Асус_АУМ03', 145000, 'Класс 3Д моделирования'),
        ('Асус_АУМ05', 125000, 'Класс 3Д моделирования')
    ]

def test_task_b2():
    db = DataBase(classes, computers, computers_classes)
    assert db.task_b2() == [
        ('Класс информатики', 125000),
        ('Класс 3Д моделирования', 125000),
        ('Класс генной инженерии', 135000)
    ]

def test_task_b3():
    db = DataBase(classes, computers, computers_classes)
    assert db.task_b3() == [
        ('Асус_АУМ03', 145000, 'Класс 3Д моделирования'),
        ('Асус_АУМ03', 145000, 'Класс 3Д моделирования
(другой)'),
        ('Асус_АУМ05', 125000, 'Класс 3Д моделирования'),
        ('Асус_АУМ05', 125000, 'Класс 3Д моделирования
(другой)'),
        ('Делл_АУМ01', 125000, 'Класс информатики'),
```

```
        ('Делл_АУМ01', 125000, 'Класс информатики (другой)'),
        ('Делл_АУМ04', 135000, 'Класс 3Д моделирования'),
        ('Делл_АУМ04', 135000, 'Класс 3Д моделирования
(другой)'),
        ('Эпл__АУМ02', 135000, 'Класс генной инженерии'),
        ('Эпл__АУМ02', 135000, 'Класс генной инженерии (другой)')
    ]
```

Пример выполнения.