Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНЖЕНЕРНЫЙ БИЗНЕС И МЕНЕДЖМЕНТ»

КАФЕДРА «ПРОМЫШЛЕННАЯ ЛОГИСТИКА» (ИБМ-3)

Отчет по выполнению рубежного контроля №2 по дисциплине “Парадигмы и конструкции языков программирования”

38.03.05 Бизнес-Информатика (уровень бакалавриата)

Студент ИБМ3-34 Шевченко Г.А.

2024 г.

Рефакторинг кода из РК№1.

```
class Department:
    def __init__(self, id, name):
        self.id = id
        self.name = name
class StudentGroup:
    def __init__(self, id, name, num_students, department_id):
        self.id = id
        self.name = name
        self.num_students = num_students
        self.department_id = department_id
class DepartmentGroup:
    def __init__(self, department_id, group_id):
        self.department_id = department_id
        self.group_id = group_id
def create_department_dict(departments):
    """Создание словаря кафедр по ID."""
    return {dept.id: dept for dept in departments}
def create_group_dict(student_groups):
    """Создание словаря групп по ID."""
    return {group.id: group for group in student_groups}
def get_sorted_dept_group_list(departments, student_groups):
    """Получение отсортированного списка всех связанных групп и кафедр."""
    department_dict = create_department_dict(departments)
    dept_group_list = [(department_dict[group.department_id], group) for group in student_groups]
    dept_group_list.sort(key=lambda x: x[0].name)
    return dept_group_list
```

```
def get_dept_student_totals(student_groups):
    """Получение общего количества студентов на каждой кафедре, отсортированного по убыванию."""
    dept_student_counts = {}
    for group in student_groups:
        dept_id = group.department_id
        dept_student_counts[dept_id] = dept_student_counts.get(dept_id, 0) + group.num_students

    return dept_student_counts
def filter_departments_with_word(departments, keyword, department_groups, student_groups_dict):
    """Фильтрация кафедр с определенным ключевым словом в названии."""
    dept_to_groups = {}
    for dg in department_groups:
        dept_to_groups.setdefault(dg.department_id, []).append(dg.group_id)

    filtered_departments = {}
    for dept_id, group_ids in dept_to_groups.items():
        dept = departments.get(dept_id)
        if keyword.lower() in dept.name.lower():
            groups = [student_groups_dict[group_id] for group_id in group_ids]
            filtered_departments[dept] = groups
    return filtered_departments
# Использование функций для выполнения запросов
departments = [
    Department(1, "Кафедра ФН"),
    Department(2, "Кафедра ИУ"),
    Department(3, "Кафедра ИБМ"),
    Department(4, "Кафедра СМ")
]
```

```
student_groups = [
    StudentGroup(1, "ФН1-13Б", 25, 1),
    StudentGroup(2, "ИУ7-24Б", 30, 2),
    StudentGroup(3, "ИБМ3-34Б", 28, 3),
    StudentGroup(4, "ИУ6-14Б", 28, 2),
    StudentGroup(5, "СМ12-43Б", 23, 4),
    StudentGroup(6,"ИБМ6-33Б",29,3),
    StudentGroup(7,"ФН3-41Б",23,1),
    StudentGroup(8,"СМ14-32Б",21,4)
]
department_groups = [
    DepartmentGroup(1, 1),
    DepartmentGroup(1, 7),
    DepartmentGroup(2, 2),
    DepartmentGroup(2, 4),
    DepartmentGroup(3, 3),
    DepartmentGroup(3, 6),
    DepartmentGroup(4, 5),
    DepartmentGroup(4, 8),
]
department_dict = create_department_dict(departments)
group_dict = create_group_dict(student_groups)
# Запрос №1
dept_group_list = get_sorted_dept_group_list(departments, student_groups)
print("Список всех связанных студенческих групп и кафедр, отсортированный по кафедрам:")
for dept, group in dept_group_list:
    print(f"Кафедра: {dept.name}, Студенческая группа: {group.name}")
```

```
# Запрос №2

dept_student_counts = get_dept_student_totals(student_groups)

dept_totals = [(department_dict[dept_id], total) for dept_id, total in dept_student_counts.items()]

dept_totals.sort(key=lambda x: x[1], reverse=True)

print("\nСписок кафедр с общим количеством студентов в студенческих группах, отсортированный по общему количеству студентов:")

for dept, total_students in dept_totals:

    print(f"Кафедра: {dept.name}, Общее количество студентов: {total_students}")

# Запрос №3

filtered_departments = filter_departments_with_word(department_dict, "кафедра", department_groups, group_dict)

print("\nСписок кафедр, у которых в названии присутствует слово «кафедра», и связанных с ними студенческих групп:")

for dept, groups in filtered_departments.items():

    print(f"Кафедра: {dept.name}")

    print("Студенческие группы:")

    for group in groups:

        print(f"- {group.name}")

    print()
```

Модульные тесты (unittest)

```
import unittest

class TestDataStructures(unittest.TestCase):

    def test_department_creation(self):

        dept = Department(1, "Кафедра ФН")

        self.assertEqual(dept.id, 1)

        self.assertEqual(dept.name, "Кафедра ФН")

    def test_student_group_creation(self):

        group = StudentGroup(1, "ФН1-13Б", 25, 1)

        self.assertEqual(group.id, 1)
```

```
        self.assertEqual(group.name, "ФН1-13Б")
        self.assertEqual(group.num_students, 25)
        self.assertEqual(group.department_id, 1)
    def test_department_group_creation(self):
        assoc = DepartmentGroup(1, 1)
        self.assertEqual(assoc.department_id, 1)
        self.assertEqual(assoc.group_id, 1)
class TestDictionaryFunctions(unittest.TestCase):
    def setUp(self):
        self.departments = [
            Department(1, "Кафедра ФН"),
            Department(2, "Кафедра ИУ"),
            Department(3, "Кафедра ИБМ"),
            Department(4, "Кафедра СМ"),
        ]
        self.student_groups = [
            StudentGroup(1, "ФН1-13Б", 25, 1),
            StudentGroup(2, "ИУ7-24Б", 30, 2),
            StudentGroup(3, "ИБМ3-34Б", 28, 3),
            StudentGroup(4, "ИУ6-14Б", 28, 2),
            StudentGroup(5, "СМ12-43Б", 23, 4),
            StudentGroup(6, "ИБМ6-33Б", 29, 3),
            StudentGroup(7, "ФН3-41Б", 23, 1),
            StudentGroup(8, "СМ14-32Б", 21, 4)
        ]
    def test_create_department_dict(self):
        department_dict = create_department_dict(self.departments)
        self.assertEqual(department_dict[1].name, "Кафедра ФН")
```

```
    def test_create_group_dict(self):
        group_dict = create_group_dict(self.student_groups)
        self.assertEqual(group_dict[2].name, "ИУ7-24Б")
class TestQueries(unittest.TestCase):
    def setUp(self):
        self.departments = [
            Department(1, "Кафедра ФН"),
            Department(2, "Кафедра ИУ"),
            Department(3, "Кафедра ИБМ"),
            Department(4, "Кафедра СМ"),
        ]
        self.student_groups = [
            StudentGroup(1, "ФН1-13Б", 25, 1),
            StudentGroup(2, "ИУ7-24Б", 30, 2),
            StudentGroup(3, "ИБМ3-34Б", 28, 3),
            StudentGroup(4, "ИУ6-14Б", 28, 2),
            StudentGroup(5, "СМ12-43Б", 23, 4),
            StudentGroup(6, "ИБМ6-33Б", 29, 3),
            StudentGroup(7, "ФН3-41Б", 23, 1),
            StudentGroup(8, "СМ14-32Б", 21, 4)
        ]
        self.department_groups = [
            DepartmentGroup(1, 1),
            DepartmentGroup(1, 7),
            DepartmentGroup(2, 2),
            DepartmentGroup(2, 4),
            DepartmentGroup(3, 3),
            DepartmentGroup(3, 6),
            DepartmentGroup(4, 5),
```

```
        DepartmentGroup(4, 8),
    ]
    self.department_dict = create_department_dict(self.departments)
    self.group_dict = create_group_dict(self.student_groups)
  def test_get_sorted_dept_group_list(self):
    dept_group_list = get_sorted_dept_group_list(self.departments, self.student_groups)
    dept_names = [dept.name for dept, _ in dept_group_list]
    self.assertEqual(dept_names, ["Кафедра ИБМ", "Кафедра ИУ", "Кафедра ИУ", "Кафедра СМ",
"Кафедра СМ", "Кафедра ФН", "Кафедра ФН"])
  def test_get_dept_student_totals(self):
    totals = get_dept_student_totals(self.student_groups)
    sorted_totals = sorted(totals.items(), key=lambda item: item[1], reverse=True)
    self.assertEqual(sorted_totals, [(2, 58), (3, 57), (1, 48), (4, 44)])
  def test_filter_departments_with_word(self):
    filtered = filter_departments_with_word(self.department_dict, "кафедра", self.department_groups,
self.group_dict)
    filtered_dept_names = [dept.name for dept in filtered]
    self.assertEqual(len(filtered), 4)
    self.assertIn("Кафедра ИБМ", filtered_dept_names)
```

Результаты тестов

Тест 1:

```
-m unittest test_module.TestDataStructures
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Тест 2:

```
-m unittest test_module.TestDictionaryFunctions
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Тест 3:

```
-m unittest test_module.TestQueries
```

```
First differing element 1:
'Кафедра ИБМ'
'Кафедра ИУ'

First list contains 1 additional elements.
First extra element 7:
'Кафедра ФН'

  ['Кафедра ИБМ',
-  'Кафедра ИБМ',
   'Кафедра ИУ',
   'Кафедра ИУ',
   'Кафедра СМ',
   'Кафедра СМ',
   'Кафедра ФН',
   'Кафедра ФН']

----------------------------------------------------------------------
Ran 3 tests in 0.025s
```

```
OK
```