

Московский государственный технический университет
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”



Дисциплина «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

Выполнил:
студент группы ИУ5-36Б
Бойко Артем Андреевич
Проверил:
Гапанюк Юрий Евгеньевич

Москва, 2025

Условия рубежного контроля №2 по курсу ПиК ЯП

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).

main.py

```
"""Основная программа для демонстрации работы системы"""
from models import Department, StudentGroup, GroupDepartment
from data_service import DataService

def create_test_data():
    """
    Создание тестовых данных

    Returns:
        Кортеж (departments, student_groups, group_departments)
    """
    # Создаем кафедры
    departments = [
        Department(1, "Алгебры и геометрии"),
        Department(2, "Вышей математики"),
        Department(3, "Аналитической химии"),
        Department(4, "Физики"),
        Department(5, "Английского языка"),
        Department(6, "Астрономии")
    ]

    # Создаем студенческие группы
    student_groups = [
        StudentGroup(1, "МАТ-101", 25, 1),
        StudentGroup(2, "МАТ-102", 30, 1),
        StudentGroup(3, "ВМ-201", 28, 2),
        StudentGroup(4, "ХИМ-301", 22, 3),
        StudentGroup(5, "АНГ-501", 35, 5),
        StudentGroup(6, "АНГ-502", 32, 5),
        StudentGroup(7, "АСТ-601", 18, 6)
    ]

    # Создаем связи многие-ко-многим
    group_departments = [
        GroupDepartment(1, 1), # МАТ-101 - Алгебры и геометрии
        GroupDepartment(2, 1), # МАТ-102 - Алгебры и геометрии
        GroupDepartment(3, 2), # ВМ-201 - Высшей математики
        GroupDepartment(4, 3), # ХИМ-301 - Аналитической химии
        GroupDepartment(5, 5), # АНГ-501 - Английского языка
```

```

        GroupDepartment(6, 5), # АНГ-502 - Английского языка
        GroupDepartment(7, 6), # АСТ-601 - Астрономии
        # Дополнительные связи для демонстрации многие-ко-многим
        GroupDepartment(1, 2), # МАТ-101 также относится к Высшей математики
        GroupDepartment(3, 1), # ВМ-201 также относится к Алгебры и геометрии
        GroupDepartment(5, 1), # АНГ-501 также относится к Алгебры и геометрии
    ]

    return departments, student_groups, group_departments

def print_test_data(departments, student_groups, group_departments):
    """Вывод тестовых данных"""
    print("=" * 80)
    print("ТЕСТОВЫЕ ДАННЫЕ")
    print("=" * 80)

    print("\nКафедры:")
    for dept in departments:
        print(f" {dept}")

    print("\nСтуденческие группы:")
    for group in student_groups:
        print(f" {group}")

    print("\nСвязи группы-кафедры:")
    for gd in group_departments:
        print(f" {gd}")

def print_query_1(service):
    """Выполнение и вывод результатов запроса №1"""
    print("\n" + "=" * 80)
    print("ЗАПРОС №1")
    print("=" * 80)
    print("Список всех кафедр, у которых название начинается с буквы «А», ")
    print("и список студенческих групп, относящихся к ним:")

    departments_with_a = service.get_departments_starting_with('A')

    for dept in departments_with_a:
        dept_groups = service.get_groups_by_department_id(dept.id)
        print(f"\nКафедра: {dept.name}")
        if dept_groups:
            for group in dept_groups:
                print(f" - Группа {group.group_name} ({group.student_count}
студентов)")
        else:
            print(" Нет студенческих групп")

def print_query_2(service):
    """Выполнение и вывод результатов запроса №2"""
    print("\n" + "=" * 80)

```

```

print("ЗАПРОС №2")
print("=" * 80)
print("Список кафедр с максимальным количеством студентов в группах")
print("в каждой кафедре, отсортированный по максимальному количеству студентов:")

department_max_students = service.get_departments_with_max_students()

for dept_name, max_students in department_max_students:
    print(f"Кафедра '{dept_name}': максимальное количество студентов = {max_students}")

def print_query_3(service):
    """Выполнение и вывод результатов запроса №3"""
    print("\n" + "=" * 80)
    print("ЗАПРОС №3")
    print("=" * 80)
    print("Список всех связанных студенческих групп и кафедр (многие-ко-многим),")
    print("отсортированный по кафедрам, сортировка по группам произвольная:")

    connections = service.get_group_department_connections()

    for department, department_groups in connections.items():
        print(f"\nКафедра: {department.name}")
        for group in department_groups:
            print(f"    - Группа {group.group_name} ({group.student_count} студентов)")

def main():
    """Основная функция программы"""
    # Создание тестовых данных
    departments, student_groups, group_departments = create_test_data()

    # Вывод тестовых данных
    print_test_data(departments, student_groups, group_departments)

    # Создание сервиса данных
    service = DataService(departments, student_groups, group_departments)

    # Выполнение и вывод результатов запросов
    print_query_1(service)
    print_query_2(service)
    print_query_3(service)

if __name__ == "__main__":
    main()

```

data_service.py

```
"""Сервис для работы с данными и выполнения запросов"""
from collections import defaultdict

class DataService:
    """Сервис для работы с данными кафедр и студенческих групп"""

    def __init__(self, departments, student_groups, group_departments):
        """
        Инициализация сервиса с данными

        Args:
            departments: список кафедр
            student_groups: список студенческих групп
            group_departments: список связей многие-ко-многим
        """
        self.departments = departments
        self.student_groups = student_groups
        self.group_departments = group_departments

        # Создаем словари для быстрого доступа
        self.group_dict = {group.id: group for group in student_groups}
        self.department_dict = {dept.id: dept for dept in departments}

    def get_departments_starting_with(self, letter):
        """
        Получить кафедры, название которых начинается с заданной буквы

        Args:
            letter: буква для поиска

        Returns:
            Список кафедр, удовлетворяющих условию
        """
        return [dept for dept in self.departments if
dept.name.startswith(letter)]

    def get_groups_by_department_id(self, department_id):
        """
        Получить студенческие группы по ID кафедры

        Args:
            department_id: ID кафедры

        Returns:
            Список студенческих групп, относящихся к кафедре
        """
        return [group for group in self.student_groups if group.department_id ==
department_id]
```

```

def get_departments_with_max_students(self):
    """
    Получить кафедры с максимальным количеством студентов в группах

    Returns:
        Список кортежей (название_кафедры,
        максимальное_количество_студентов),
        отсортированный по убыванию количества студентов
    """
    # Группируем студенческие группы по кафедрам
    groups_by_department = defaultdict(list)
    for group in self.student_groups:
        groups_by_department[group.department_id].append(group)

    # Находим максимальное количество студентов для каждой кафедры
    department_max_students = []
    for dept_id, dept_groups in groups_by_department.items():
        if dept_groups:
            max_students = max(group.student_count for group in dept_groups)
            dept_name = next(dept.name for dept in self.departments if
            dept.id == dept_id)
            department_max_students.append((dept_name, max_students))

    # Сортируем по максимальному количеству студентов (по убыванию)
    department_max_students.sort(key=lambda x: x[1], reverse=True)

    return department_max_students

def get_group_department_connections(self):
    """
    Получить все связи группы-кафедры, отсортированные по кафедрам

    Returns:
        Словарь, где ключ - кафедра, значение - список связанных групп
    """
    connections_by_department = defaultdict(list)

    for gd in self.group_departments:
        group = self.group_dict.get(gd.group_id)
        department = self.department_dict.get(gd.department_id)
        if group and department:
            connections_by_department[department].append(group)

    # Сортируем кафедры по названию
    sorted_connections = {}
    for department in sorted(connections_by_department.keys(), key=lambda x:
    x.name):
        sorted_connections[department] =
        connections_by_department[department]

    return sorted_connections

```

models.py

```
"""Модели данных для системы кафедр и студенческих групп"""
```

```
class Department:
```

```
    """Класс Кафедра"""
```

```
    def __init__(self, id, name):
```

```
        self.id = id
```

```
        self.name = name
```

```
    def __repr__(self):
```

```
        return f"Department(id={self.id}, name='{self.name}')
```

```
class StudentGroup:
```

```
    """Класс Студенческая группа"""
```

```
    def __init__(self, id, group_name, student_count, department_id):
```

```
        self.id = id
```

```
        self.group_name = group_name
```

```
        self.student_count = student_count
```

```
        self.department_id = department_id
```

```
    def __repr__(self):
```

```
        return f"StudentGroup(id={self.id}, group_name='{self.group_name}',
```

```
student_count={self.student_count}, department_id={self.department_id})"
```

```
class GroupDepartment:
```

```
    """Класс для связи многие-ко-многим между Студенческой группой и Кафедрой"""
```

```
    def __init__(self, group_id, department_id):
```

```
        self.group_id = group_id
```

```
        self.department_id = department_id
```

```
    def __repr__(self):
```

```
        return f"GroupDepartment(group_id={self.group_id},
```

```
department_id={self.department_id})"
```

test_data_service.py

```
"""Модульные тесты для сервиса данных"""
```

```
import unittest
```

```
from models import Department, StudentGroup, GroupDepartment
```

```
from data_service import DataService
```

```
class TestDataService(unittest.TestCase):
```

```
    """Класс тестов для DataService"""
```

```
    def setUp(self):
```

```
        """Настройка тестовых данных перед каждым тестом"""
```

```
        # Создаем тестовые данные (ТОЛЬКО для тестов, без лишних данных из
```

```
main.py)
```

```
        self.departments = [
```

```

        Department(1, "Алгебры и геометрии"),
        Department(2, "Высшей математики"),
        Department(3, "Аналитической химии"),
        Department(4, "Физики"),
        Department(5, "Английского языка"),
    ]

    self.student_groups = [
        StudentGroup(1, "МАТ-101", 25, 1),
        StudentGroup(2, "МАТ-102", 30, 1),
        StudentGroup(3, "ВМ-201", 28, 2),
        StudentGroup(4, "ХИМ-301", 22, 3),
        StudentGroup(5, "АНГ-501", 35, 5),
    ]

    self.group_departments = [
        GroupDepartment(1, 1),
        GroupDepartment(2, 1),
        GroupDepartment(3, 2),
        GroupDepartment(4, 3),
        GroupDepartment(5, 5),
        GroupDepartment(1, 2), # Дополнительная связь
    ]

    # Создаем экземпляр сервиса
    self.service = DataService(
        self.departments,
        self.student_groups,
        self.group_departments
    )

def test_get_departments_starting_with(self):
    """Тест для метода get_departments_starting_with"""
    # Тест 1: Поиск кафедр, начинающихся на 'А'
    result = self.service.get_departments_starting_with('А')
    self.assertEqual(len(result), 3) # Исправлено: должно быть 3 кафедры
    # Проверяем конкретные кафедры
    dept_names = [dept.name for dept in result]
    expected_names = ["Алгебры и геометрии", "Аналитической химии",
"Английского языка"]
    self.assertEqual(dept_names, expected_names) # Проверяем содержимое
без учета порядка

    # Тест 2: Поиск кафедр, начинающихся на 'В'
    result = self.service.get_departments_starting_with('В')
    self.assertEqual(len(result), 1) # Должна быть 1 кафедра
    self.assertEqual(result[0].name, "Высшей математики")

    # Тест 3: Поиск кафедр, начинающихся на 'Ф'
    result = self.service.get_departments_starting_with('Ф')
    self.assertEqual(len(result), 1) # Должна быть 1 кафедра

```



```

self.assertEqual(result[0].name, "Физики")

# Тест 4: Поиск кафедр, начинающихся на 'X' (нет таких)
result = self.service.get_departments_starting_with('X')
self.assertEqual(len(result), 0) # Не должно быть кафедр

def test_get_groups_by_department_id(self):
    """Тест для метода get_groups_by_department_id"""
    # Тест 1: Группы кафедры с ID=1
    result = self.service.get_groups_by_department_id(1)
    self.assertEqual(len(result), 2) # Должно быть 2 группы
    group_names = [group.group_name for group in result]
    self.assertEqual(group_names, ["MAT-101", "MAT-102"])

    # Тест 2: Группы кафедры с ID=2
    result = self.service.get_groups_by_department_id(2)
    self.assertEqual(len(result), 1) # Должна быть 1 группа
    self.assertEqual(result[0].group_name, "BM-201")

    # Тест 3: Группы кафедры с ID=5
    result = self.service.get_groups_by_department_id(5)
    self.assertEqual(len(result), 1) # Должна быть 1 группа
    self.assertEqual(result[0].group_name, "АНГ-501")

    # Тест 4: Группы кафедры с ID=999 (не существует)
    result = self.service.get_groups_by_department_id(999)
    self.assertEqual(len(result), 0) # Не должно быть групп

def test_get_departments_with_max_students(self):
    """Тест для метода get_departments_with_max_students"""
    result = self.service.get_departments_with_max_students()

    # Проверяем количество результатов
    self.assertEqual(len(result), 4) # Должно быть 4 кафедры (у кафедры 4
нет групп)

    # Проверяем правильность сортировки (по убыванию количества студентов)
    self.assertEqual(result[0][1], 35) # Максимум у кафедры английского
    self.assertEqual(result[1][1], 30) # Затем у алгебры
    self.assertEqual(result[2][1], 28) # Затем у высшей математики
    self.assertEqual(result[3][1], 22) # Затем у химии

    # Проверяем названия кафедр
    self.assertEqual(result[0][0], "Английского языка")
    self.assertEqual(result[1][0], "Алгебры и геометрии")
    self.assertEqual(result[2][0], "Высшей математики")
    self.assertEqual(result[3][0], "Аналитической химии")

    # Проверяем структуру результата
    for dept_name, max_students in result:
        self.assertIsInstance(dept_name, str)

```

```

        self.assertIsInstance(max_students, int)
        self.assertGreater(max_students, 0)

def test_get_group_department_connections(self):
    """Тест для метода get_group_department_connections"""
    result = self.service.get_group_department_connections()

    # Проверяем количество кафедр в результате
    self.assertEqual(len(result), 4) # Должно быть 4 кафедры (у кафедры 4
нет связей)

    # Проверяем сортировку по названиям кафедр
    departments = list(result.keys())
    dept_names = [dept.name for dept in departments]
    expected_names = ["Алгебры и геометрии", "Аналитической химии",
                      "Английского языка", "Вышей математики"]
    self.assertEqual(dept_names, expected_names)

    # Проверяем связи для кафедры "Алгебры и геометрии"
    algebra_groups = result[departments[0]]
    self.assertEqual(len(algebra_groups), 2) # Исправлено: 2 группы связаны
с этой кафедрой
    group_names = sorted([g.group_name for g in algebra_groups])
    self.assertEqual(group_names, ["MAT-101", "MAT-102"]) # Только эти 2
группы

    # Проверяем связи для кафедры "Вышей математики"
    math_groups = result[departments[3]]
    self.assertEqual(len(math_groups), 2) # 2 группы: BM-201 и MAT-101
(через дополнительную связь)
    group_names = sorted([g.group_name for g in math_groups])
    self.assertEqual(group_names, ["BM-201", "MAT-101"])

if __name__ == '__main__':
    # Запуск тестов с детальным выводом
    suite = unittest.TestLoader().loadTestsFromTestCase(TestDataService)
    runner = unittest.TextTestRunner(verbosity=2)
    runner.run(suite)

```

Результаты выполнения программ main.py

● @artem-hedgehog → /workspaces/ProgLangParadigms_BoikoA_2025/RK_2 (main) \$ python main.py

```
=====
ТЕСТОВЫЕ ДАННЫЕ
=====
```

Кафедры:

```
Department(id=1, name='Алгебры и геометрии')
Department(id=2, name='Высшей математики')
Department(id=3, name='Аналитической химии')
Department(id=4, name='Физики')
Department(id=5, name='Английского языка')
Department(id=6, name='Астрономии')
```

Студенческие группы:

```
StudentGroup(id=1, group_name='МАТ-101', student_count=25, department_id=1)
StudentGroup(id=2, group_name='МАТ-102', student_count=30, department_id=1)
StudentGroup(id=3, group_name='ВМ-201', student_count=28, department_id=2)
StudentGroup(id=4, group_name='ХИМ-301', student_count=22, department_id=3)
StudentGroup(id=5, group_name='АНГ-501', student_count=35, department_id=5)
StudentGroup(id=6, group_name='АНГ-502', student_count=32, department_id=5)
StudentGroup(id=7, group_name='АСТ-601', student_count=18, department_id=6)
```

Связи группы-кафедры:

```
GroupDepartment(group_id=1, department_id=1)
GroupDepartment(group_id=2, department_id=1)
GroupDepartment(group_id=3, department_id=2)
GroupDepartment(group_id=4, department_id=3)
GroupDepartment(group_id=5, department_id=5)
GroupDepartment(group_id=6, department_id=5)
GroupDepartment(group_id=7, department_id=6)
GroupDepartment(group_id=1, department_id=2)
GroupDepartment(group_id=3, department_id=1)
GroupDepartment(group_id=5, department_id=1)
```

=====

ЗАПРОС №1

=====

Список всех кафедр, у которых название начинается с буквы «А»,
и список студенческих групп, относящихся к ним:

Кафедра: Алгебры и геометрии

- Группа МАТ-101 (25 студентов)
- Группа МАТ-102 (30 студентов)

Кафедра: Аналитической химии

- Группа ХИМ-301 (22 студентов)

Кафедра: Английского языка

- Группа АНГ-501 (35 студентов)
- Группа АНГ-502 (32 студентов)

Кафедра: Астрономии

- Группа АСТ-601 (18 студентов)

=====

ЗАПРОС №2

=====

Список кафедр с максимальным количеством студентов в группах
в каждой кафедре, отсортированный по максимальному количеству студентов:

Кафедра 'Английского языка': максимальное количество студентов = 35

Кафедра 'Алгебры и геометрии': максимальное количество студентов = 30

Кафедра 'Высшей математики': максимальное количество студентов = 28

Кафедра 'Аналитической химии': максимальное количество студентов = 22

Кафедра 'Астрономии': максимальное количество студентов = 18

```
=====
ЗАПРОС №3
=====
```

Список всех связанных студенческих групп и кафедр (многие-ко-многим),
отсортированный по кафедрам, сортировка по группам произвольная:

Кафедра: Алгебры и геометрии

- Группа МАТ-101 (25 студентов)
- Группа МАТ-102 (30 студентов)
- Группа ВМ-201 (28 студентов)
- Группа АНГ-501 (35 студентов)

Кафедра: Аналитической химии

- Группа ХИМ-301 (22 студентов)

Кафедра: Английского языка

- Группа АНГ-501 (35 студентов)
- Группа АНГ-502 (32 студентов)

Кафедра: Астрономии

- Группа АСТ-601 (18 студентов)

Кафедра: Высшей математики

- Группа ВМ-201 (28 студентов)
- Группа МАТ-101 (25 студентов)

test_data_service.py

```
● @artem-hedgehog → /workspaces/ProgLangParadigms_BoikoA_2025/RK_2 (main) $ python test_data_service.py
test_get_departments_starting_with (__main__.TestDataService.test_get_departments_starting_with)
Тест для метода get_departments_starting_with ... ok
test_get_departments_with_max_students (__main__.TestDataService.test_get_departments_with_max_students)
Тест для метода get_departments_with_max_students ... ok
test_get_group_department_connections (__main__.TestDataService.test_get_group_department_connections)
Тест для метода get_group_department_connections ... ok
test_get_groups_by_department_id (__main__.TestDataService.test_get_groups_by_department_id)
Тест для метода get_groups_by_department_id ... ok

-----
Ran 4 tests in 0.001s

OK
```