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
“Разработка тестов на языке Python”

Выполнил:
Студент группы ИУ5-34Б
Невечеря А. Д.
Преподаватель:
Гапанюк Ю. Е.

Москва 2025

Код rk2.py

```
from typing import List, Tuple, Dict
from dataclasses import dataclass

@dataclass
class Course:
    id: int
    name: str

@dataclass
class Teacher:
    id: int
    surname: str
    salary: int
    course_id: int

@dataclass
class TeacherCourse:
    teacher_id: int
    course_id: int

class UniversitySystem:
    def __init__(self):
        self.courses: List[Course] = []
        self.teachers: List[Teacher] = []
        self.teacher_courses: List[TeacherCourse] = []
        self._course_by_id: Dict[int, Course] = {}
        self._teacher_by_id: Dict[int, Teacher] = {}

    def load_data(self,
                  courses: List[Course],
                  teachers: List[Teacher],
                  teacher_courses: List[TeacherCourse]):
        """Загрузить данные в систему"""
        self.courses = courses
        self.teachers = teachers
        self.teacher_courses = teacher_courses
        self._course_by_id = {c.id: c for c in courses}
        self._teacher_by_id = {t.id: t for t in teachers}

    def query_1_teachers_starting_with_a(self) -> List[Tuple[str, str]]:
        """
```

```
        B1: Фамилии преподавателей, начинающиеся с буквы «А»,
        и названия курсов, которые они ведут
        """
        pairs = [
            (t.surname, self._course_by_id[t.course_id].name)
            for t in self.teachers
            if t.surname.upper().startswith("А")
        ]
        return sorted(pairs, key=lambda x: (x[0], x[1]))

    def query_2_min_salary_per_course(self) -> List[Tuple[str, int]]:
        """
        B2: Название курса и минимальная зарплата преподавателей,
        которые ведут этот курс (в основном курсе)
        """
        groups = {}
        for t in self.teachers:
            groups.setdefault(t.course_id, []).append(t.salary)

        agg = [(self._course_by_id[cid].name, min(salaries))
               for cid, salaries in groups.items()]
        return sorted(agg, key=lambda x: x[1])

    def query_3_all_teacher_course_m2m(self) -> List[Tuple[str, str]]:
        """
        B3: Связь многие-ко-многим: все преподаватели и все курсы,
        которые они ведут (по таблице связи)
        """
        pairs = [
            (self._teacher_by_id[link.teacher_id].surname,
             self._course_by_id[link.course_id].name)
            for link in self.teacher_courses
        ]
        return sorted(pairs, key=lambda x: x[0])
def create_sample_data() -> tuple:
    courses = [
        Course(1, "Математический анализ"),
        Course(2, "Физика"),
        Course(3, "Программирование"),
    ]

    teachers = [
        Teacher(1, "Абрамов", 50000, 1),
        Teacher(2, "Антонов", 60000, 1),
        Teacher(3, "Борисов", 55000, 2),
```

```
        Teacher(4, "Алексеев", 45000, 3),
        Teacher(5, "Смирнов", 70000, 3),
    ]

    teacher_courses = [
        TeacherCourse(1, 1),
        TeacherCourse(2, 1),
        TeacherCourse(3, 2),
        TeacherCourse(4, 3),
        TeacherCourse(5, 3),
        TeacherCourse(1, 3),
    ]

    return courses, teachers, teacher_courses

def main():
    system = UniversitySystem()
    courses, teachers, teacher_courses = create_sample_data()
    system.load_data(courses, teachers, teacher_courses)
    q1 = system.query_1_teachers_starting_with_a()
    q2 = system.query_2_min_salary_per_course()
    q3 = system.query_3_all_teacher_course_m2m()
    print("Задание B1: Преподаватели на 'A' и их курсы")
    for surname, course_name in q1:
        print(f"   - {surname} — {course_name}")

    print("\nЗадание B2: Минимальная зарплата по курсам")
    for course_name, min_salary in q2:
        print(f"   - {course_name}: {min_salary} руб.")

    print("\nЗадание B3: Все связи преподаватель-курс")
    for surname, course_name in q3:
        print(f"   - {surname} — {course_name}")

if __name__ == "__main__":
    main()
```

Код test.py

```
import unittest
from rk2 import UniversitySystem, Course, Teacher, TeacherCourse

class TestUniversitySystem(unittest.TestCase):
    def setUp(self):
        self.system = UniversitySystem()
        self.courses = [
            Course(1, "Математический анализ"),
            Course(2, "Физика"),
            Course(3, "Программирование"),
        ]

        self.teachers = [
            Teacher(1, "Абрамов", 50000, 1),
            Teacher(2, "Антонов", 60000, 1),
            Teacher(3, "Борисов", 55000, 2),
            Teacher(4, "Алексеев", 45000, 3),
            Teacher(5, "Смирнов", 70000, 3),
        ]

        self.teacher_courses = [
            TeacherCourse(1, 1),
            TeacherCourse(2, 1),
            TeacherCourse(3, 2),
            TeacherCourse(4, 3),
            TeacherCourse(5, 3),
            TeacherCourse(1, 3),
        ]
        self.system.load_data(self.courses, self.teachers,
self.teacher_courses)

    def test_query_1_teachers_starting_with_a(self):
        result = self.system.query_1_teachers_starting_with_a()
        expected = [
            ("Абрамов", "Математический анализ"),
            ("Алексеев", "Программирование"),
            ("Антонов", "Математический анализ"),
        ]
        self.assertEqual(len(result), 3, "Должно быть 3 преподавателя на
'А'")
        self.assertListEqual(result, expected, "Результаты не совпадают с
ожидаемыми")
```

```
        for surname, _ in result:
            self.assertTrue(surname.startswith('А'), f"Фамилия {surname}
должна начинаться с 'А'")

    def test_query_2_min_salary_per_course(self):
        result = self.system.query_2_min_salary_per_course()
        expected = [
            ("Программирование", 45000),
            ("Физика", 55000),
            ("Математический анализ", 50000),
        ]
        self.assertEqual(len(result), 3, "Должно быть 3 курса")
        self.assertEqual(set(result), set(expected), "Минимальные
зарплаты не совпадают")
        salaries = [salary for _, salary in result]
        self.assertEqual(salaries, sorted(salaries), "Результаты должны
быть отсортированы по зарплате")

    def test_query_3_all_teacher_course_m2m(self):
        result = self.system.query_3_all_teacher_course_m2m()

        expected = [
            ("Абрамов", "Математический анализ"),
            ("Абрамов", "Программирование"),
            ("Алексеев", "Программирование"),
            ("Антонов", "Математический анализ"),
            ("Борисов", "Физика"),
            ("Смирнов", "Программирование"),
        ]

        self.assertEqual(len(result), 6, f"Должно быть 6 связей, получено
{len(result)}")

        self.assertListEqual(result, expected, "Связи не совпадают с
ожидаемыми")

        surnames = [surname for surname, _ in result]
        self.assertEqual(surnames, sorted(surnames), "Результаты должны
быть отсортированы по фамилии")

        abramov_count = sum(1 for surname, _ in result if surname ==
"Абрамов")
        self.assertEqual(abramov_count, 2, "У Абрамова должно быть 2
связи")
```

```
    def test_empty_data(self):
        empty_system = UniversitySystem()
        empty_system.load_data([], [], [])
        self.assertEqual(empty_system.query_1_teachers_starting_with_a(),
[])
        self.assertEqual(empty_system.query_2_min_salary_per_course(),
[])
        self.assertEqual(empty_system.query_3_all_teacher_course_m2m(),
[])

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