Московский государственный технический университет им. Н. Э. Баумана

Факультет «Радиотехнический»
Кафедра «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №1

Выполнил:
студент группы РТ5-31Б
Бондарь М. В.

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Москва, 2025 г.

Текст программы

```python
import unittest
from operator import itemgetter

class Driver:
    """Водитель"""
    def __init__(self, id, fio, salary, park_id):
        self.id = id
        self.fio = fio
        self.salary = salary
        self.park_id = park_id

class Autopark:
    """Автопарк"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class DriverAutopark:
    """
    'Водители автопарка' для реализации
    связи многие-ко-многим
    """
    def __init__(self, park_id, driver_id):
        self.park_id = park_id
        self.driver_id = driver_id

# Данные вынесены глобально для доступа из тестов и main
autoparks = [
    Autopark(1, 'Городской Автопарк'),
    Autopark(2, 'Логистик Транс'),
    Autopark(3, 'Сити Мобил Парк'),
]

drivers = [
    Driver(1, 'Алексеев', 50000, 1),
    Driver(2, 'Петров', 80000, 2),
    Driver(3, 'Сидоров', 45000, 2),
    Driver(4, 'Иванов', 35000, 3),
    Driver(5, 'Смирнов', 40000, 3),
]

drivers_autoparks = [
    DriverAutopark(1, 1),
    DriverAutopark(2, 2),
    DriverAutopark(2, 3),
    DriverAutopark(3, 4),
    DriverAutopark(3, 5),
    DriverAutopark(3, 2),
]
```

```python
# --- ФУНКЦИИ ЗАПРОСОВ (Рефакторинг) ---

def get_parks_with_keyword(autoparks, drivers, keyword):
    """
    Запрос 1: Автопарки с ключевым словом в названии и их водители
(1:M).
    """
    result = {}
    for p in autoparks:
        if keyword in p.name:
            d_s = [d.fio for d in drivers if d.park_id == p.id]
            result[p.name] = d_s
    return result

def get_avg_salary_by_park(autoparks, drivers):
    """
    Запрос 2: Средняя зарплата по паркам, сортировка (1:M).
    """
    # Сначала строим связь 1:M
    one_to_many = [(d.fio, d.salary, p.name)
                   for p in autoparks
                   for d in drivers
                   if d.park_id == p.id]

    res_unsorted = []
    for p in autoparks:
        d_salaries = list(filter(lambda i: i[2] == p.name, one_to_many))
        if len(d_salaries) > 0:
            sals = [salary for _, salary, _ in d_salaries]
            avg_salary = round(sum(sals) / len(sals), 2)
            res_unsorted.append((p.name, avg_salary))

    return sorted(res_unsorted, key=itemgetter(1), reverse=True)

def get_drivers_by_start_letter(drivers, autoparks, drivers_autoparks,
letter):
    """
    Запрос 3: Водители на букву 'letter' и их парки (M:M).
    """
    # Соединение данных многие-ко-многим
    many_to_many_temp = [(p.name, dp.park_id, dp.driver_id)
                         for p in autoparks
                         for dp in drivers_autoparks
                         if p.id == dp.park_id]

    many_to_many = [(d.fio, d.salary, park_name)
                    for park_name, park_id, driver_id in
many_to_many_temp
                    for d in drivers if d.id == driver_id]

    result = {}
    target_drivers = [d for d in drivers if d.fio.startswith(letter)]
```

```python
    for d in target_drivers:
        d_parks = list(filter(lambda i: i[0] == d.fio, many_to_many))
        d_parks_names = [x for _,_,x in d_parks]
        result[d.fio] = d_parks_names

    return result


def main():
    """Основная функция - вывод результатов"""
    print('Задание Е1')
    res1 = get_parks_with_keyword(autoparks, drivers, 'Парк')
    for park, workers in res1.items():
        print(f'{park}: {", ".join(workers)}')

    print('\nЗадание Е2')
    res2 = get_avg_salary_by_park(autoparks, drivers)
    print(res2)

    print('\nЗадание Е3')
    res3 = get_drivers_by_start_letter(drivers, autoparks,
drivers_autoparks, 'П')
    print(res3)


# --- МОДУЛЬНЫЕ ТЕСТЫ (TDD) ---

class TestRk2(unittest.TestCase):

    def test_get_parks_with_keyword(self):
        # Проверяем, что находится "Городской Автопарк" по слову "Город"
        res = get_parks_with_keyword(autoparks, drivers, 'Город')
        self.assertIn('Городской Автопарк', res)
        # Проверяем, что список водителей верен (Алексеев id=1)
        self.assertEqual(res['Городской Автопарк'], ['Алексеев'])

    def test_get_avg_salary_by_park(self):
        # В 'Логистик Транс' (id=2) работают Петров (80000) и Сидоров
(45000)
        # Среднее = (80000 + 45000) / 2 = 62500.0
        res = get_avg_salary_by_park(autoparks, drivers)
        # Ищем кортеж с Логистик Транс
        park_stat = next(item for item in res if item[0] == 'Логистик
Транс')
        self.assertEqual(park_stat[1], 62500.0)
        # Проверка сортировки (первый элемент должен быть самым большим)
        self.assertTrue(res[0][1] >= res[1][1])

    def test_get_drivers_by_start_letter(self):
        # Петров работает в 2 парках (Логистик и Сити) по данным M:M
        res = get_drivers_by_start_letter(drivers, autoparks,
```

```
drivers_autoparks, 'П')
        self.assertIn('Петров', res)
        self.assertEqual(len(res['Петров']), 2)
        self.assertIn('Логистик Транс', res['Петров'])
        self.assertIn('Сити Мобил Парк', res['Петров'])

    def test_data_integrity(self):
        """Проверка корректности связей"""
        park_ids = [p.id for p in autoparks]
        for d in drivers:
            self.assertTrue(d.park_id in park_ids, f"Водитель {d.fio}
ссылается на несуществующий парк {d.park_id}")

if __name__ == '__main__':
    main()
    # Запуск тестов
    print("\n--- Запуск тестов ---")
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

Вывод программы

Ran 4 tests in 0.003s

OK

Process finished with exit code 0

Проверка работы тестов при некорректных данных (привязка водителя к несуществующему парку):

```
drivers = [
    Driver(1, 'Алексеев', 50000, 1),
    Driver(2, 'Петров', 80000, 2),
    Driver(3, 'Сидоров', 45000, 2),
    Driver(4, 'Иванов', 35000, 50),
    Driver(5, 'Смирнов', 40000, 3),
]
```

```
Failure
Traceback (most recent call last):
  File "C:\Users\Matthew\PycharmProjects\rk 1-2\rk2.py", line 158, in test_data_integrity
    self.assertTrue(d.park_id in park_ids, f"Водитель {d.fio} ссылается на несуществующий парк {d.park_id}")
~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

AssertionError: False is not true : Водитель Иванов ссылается на несуществующий парк 50

Проверка работы тестов при ошибке в логике программы (функция поиска возвращает неверный результат):

```
for p in autoparks:
    if keyword in p.name and False:
        d_s = [d.fio for d in drivers if d.park_id == p.id]
        result[p.name] = d_s
return result
```

Failure
Traceback (most recent call last):
  File "C:\Users\Matthew\PycharmProjects\rk 1-2\rk2.py", line 132, in test_get_parks_with_keyword
    self.assertIn('Городской Автопарк', res)
    ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: 'Городской Автопарк' not found in {}
    ~~~~~~~~~~~~~~