

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 ИУ5-35Б

Александров Г.С.

Преподаватель:

Гапанюк Ю. Е.

Москва 2025

Задание

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).

Листинг кода

Файл autopark_system.py:

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

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

class DriverAutopark:
    def __init__(self, park_id, driver_id):
        self.park_id = park_id
        self.driver_id = driver_id

class AutoparkSystem:

    def __init__(self):
        self.autoparks = [
            Autopark(1, 'Автопарк Северный'),
            Autopark(2, 'Автобусный парк'),
            Autopark(3, 'Таксопарк Центральный'),
            Autopark(4, 'Аэропортный автопарк'),
            Autopark(5, 'Грузовой парк'),
        ]

        self.drivers = [
            Driver(1, 'Иванов', 50000, 1),
            Driver(2, 'Петров', 45000, 2),
            Driver(3, 'Сидоров', 55000, 3),
            Driver(4, 'Алексеев', 48000, 3),
            Driver(5, 'Антонов', 52000, 4),
            Driver(6, 'Александров', 47000, 5),
        ]

        self.drivers_autoparks = [
            DriverAutopark(1, 1),
            DriverAutopark(2, 2),
```

```

        DriverAutopark(3, 3),
        DriverAutopark(3, 4),
        DriverAutopark(4, 5),
        DriverAutopark(5, 6),
        DriverAutopark(1, 3),
        DriverAutopark(2, 4),
    ]

```

```

def get_one_to_many(self):
    return [(d.fio, d.salary, p.name)
            for p in self.autoparks
            for d in self.drivers
            if d.park_id == p.id]

def get_many_to_many(self):
    many_to_many_temp = [(p.name, da.park_id, da.driver_id)
                          for p in self.autoparks
                          for da in self.drivers_autoparks
                          if p.id == da.park_id]

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

def task_g1(self):
    one_to_many = self.get_one_to_many()
    result = {}

    for p in self.autoparks:
        if p.name.startswith('A'):
            p_drivers = list(filter(lambda i: i[2] == p.name, one_to_many))
            p_drivers_names = [x for x, _, _ in p_drivers]
            result[p.name] = p_drivers_names

    return result

def task_g2(self):
    one_to_many = self.get_one_to_many()
    park_drivers_dict = {}

    for driver_fio, salary, park_name in one_to_many:
        if park_name not in park_drivers_dict:
            park_drivers_dict[park_name] = []
        park_drivers_dict[park_name].append(salary)

```

```

        result_unsorted = []
        for park_name, salaries in park_drivers_dict.items():
            max_salary = max(salaries)
            result_unsorted.append((park_name, max_salary))

        return sorted(result_unsorted, key=lambda x: x[1], reverse=True)

def task_g3(self):
    many_to_many = self.get_many_to_many()
    return sorted(many_to_many, key=lambda x: x[2])

def main():
    system = AutoparkSystem()

    print('Задание Г1')
    res_g1 = system.task_g1()
    print("Автопарки, начинающиеся на 'А', и их водители:")
    for park, drivers_list in res_g1.items():
        print(f"{park}: {drivers_list}")

    print('\nЗадание Г2')
    res_g2 = system.task_g2()
    print("Автопарки с максимальной зарплатой водителей:")
    for park, max_sal in res_g2:
        print(f"{park}: {max_sal} руб.")

    print('\nЗадание Г3')
    res_g3 = system.task_g3()
    print("Все связанные водители и автопарки:")
    for driver, salary, park in res_g3:
        print(f"{park}: {driver} - {salary} руб.")

if __name__ == '__main__':
    main()

```

Файл test_autopark_system.py

```

import unittest
from autopark_system import AutoparkSystem

class TestAutoparkSystem(unittest.TestCase):

    def setUp(self):

```

```

self.system = AutoparkSystem()

def test_task_g1_autoparks_starting_with_A(self):
    result = self.system.task_g1()

    self.assertIsInstance(result, dict)

    expected_parks = ['Автопарк Северный', 'Аэропортный автопарк']
    for park in expected_parks:
        self.assertIn(park, result)

    self.assertEqual(result['Автопарк Северный'], ['Иванов'])
    self.assertEqual(result['Аэропортный автопарк'], ['Антонов'])

    self.assertNotIn('Таксопарк Центральный', result)
    self.assertNotIn('Грузовой парк', result)

def test_task_g2_max_salary_per_autopark(self):
    result = self.system.task_g2()

    self.assertIsInstance(result, list)

    for item in result:
        self.assertIsInstance(item, tuple)
        self.assertEqual(len(item), 2)
        self.assertIsInstance(item[0], str)
        self.assertIsInstance(item[1], int)

    salaries = [item[1] for item in result]
    self.assertEqual(salaries, sorted(salaries, reverse=True))

    result_dict = dict(result)
    self.assertEqual(result_dict['Таксопарк Центральный'], 55000)
    self.assertEqual(result_dict['Аэропортный автопарк'], 52000)
    self.assertEqual(result_dict['Грузовой парк'], 47000)

def test_task_g3_all_connections_sorted(self):
    result = self.system.task_g3()

    self.assertIsInstance(result, list)
    self.assertGreater(len(result), 0)

    for item in result:
        self.assertIsInstance(item, tuple)

```

```

        self.assertEqual(len(item), 3)
        self.assertIsInstance(item[0], str)
        self.assertIsInstance(item[1], int)
        self.assertIsInstance(item[2], str)

    autopark_names = [item[2] for item in result]
    self.assertEqual(autopark_names, sorted(autopark_names))

    drivers_by_park = {}
    for driver, salary, park in result:
        if park not in drivers_by_park:
            drivers_by_park[park] = []
        drivers_by_park[park].append(driver)

    self.assertIn('Иванов', drivers_by_park.get('Автобусный парк', []))
    self.assertIn('Сидоров', drivers_by_park.get('Автопарк Северный', []))
    self.assertIn('Алексеев', drivers_by_park.get('Автобусный парк', []))

def test_get_one_to_many_connections(self):
    connections = self.system.get_one_to_many()

    self.assertIsInstance(connections, list)
    self.assertGreater(len(connections), 0)

    for connection in connections:
        self.assertEqual(len(connection), 3)
        self.assertIsInstance(connection[0], str)
        self.assertIsInstance(connection[1], int)
        self.assertIsInstance(connection[2], str)

def test_get_many_to_many_connections(self):
    connections = self.system.get_many_to_many()

    self.assertIsInstance(connections, list)
    self.assertGreater(len(connections), 0)

    for connection in connections:
        self.assertEqual(len(connection), 3)
        self.assertIsInstance(connection[0], str)
        self.assertIsInstance(connection[1], int)
        self.assertIsInstance(connection[2], str)

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

```