Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

«**Вариант Д, 20**»

Выполнил
Студент группы ИУ5-31Б
Ван
Чжувэнь

Проверил:
Гапанюк Ю.Е.

2025 г.

Листинг кода

rk1_refactored.py

```
# rk1_refactored.py
from operator import itemgetter

class Detail:
    """Деталь"""
    def __init__(self, id, name, cost, supplier_id):
        self.id = id
        self.name = name
        self.cost = cost
        self.supplier_id = supplier_id

class Supplier:
    """Поставщик"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class DetailSupplier:
    """Связь многие-ко-многим"""
    def __init__(self, supplier_id, detail_id):
        self.supplier_id = supplier_id
        self.detail_id = detail_id

class DataService:
    """Сервис для работы с данными"""

    @staticmethod
    def create_test_data():
        """Создание тестовых данных"""
        suppliers = [
            Supplier(1, 'Авангард'),
            Supplier(2, 'Бетакон'),
            Supplier(3, 'Агропром'),
            Supplier(4, 'Металлинвест'),
        ]

        details = [
            Detail(1, 'Болтов', 1500, 1),
            Detail(2, 'Винтов', 2000, 2),
            Detail(3, 'Шурупов', 1200, 1),
            Detail(4, 'Гайков', 1800, 3),
            Detail(5, 'Резьбов', 2200, 4),
        ]

        detail_suppliers = [
            DetailSupplier(1, 1),
            DetailSupplier(1, 3),
            DetailSupplier(2, 2),
```

```
            DetailSupplier(3, 4),
            DetailSupplier(4, 5),
        ]

        return suppliers, details, detail_suppliers

    @staticmethod
    def create_one_to_many(suppliers, details):
        """Создание связи один-ко-многим"""
        return [(d.name, d.cost, s.name)
                for s in suppliers
                for d in details
                if d.supplier_id == s.id]

    @staticmethod
    def create_many_to_many(suppliers, details, detail_suppliers):
        """Создание связи многие-ко-многим"""
        many_to_many_temp = [(s.name, ds.supplier_id, ds.detail_id)
                             for s in suppliers
                             for ds in detail_suppliers
                             if s.id == ds.supplier_id]

        return [(d.name, d.cost, supplier_name)
                for supplier_name, supplier_id, detail_id in many_to_many_temp
                for d in details if d.id == detail_id]

class QueryService:
    """Сервис для выполнения запросов"""

    @staticmethod
    def query1(one_to_many):
        """Запрос 1: Детали с названием на 'ов'"""
        return list(filter(lambda i: i[0].endswith('ов'), one_to_many))

    @staticmethod
    def query2(one_to_many, suppliers):
        """Запрос 2: Средняя стоимость деталей по поставщикам"""
        res_unsorted = []
        for s in suppliers:
            s_details = list(filter(lambda i: i[2] == s.name, one_to_many))
            if len(s_details) > 0:
                s_costs = [cost for _, cost, _ in s_details]
                avg_cost = round(sum(s_costs) / len(s_costs), 2)
                res_unsorted.append((s.name, avg_cost))
        return sorted(res_unsorted, key=itemgetter(1))

    @staticmethod
    def query3(many_to_many, suppliers):
        """Запрос 3: Поставщики на 'А' и их детали"""
        res = {}
        for s in suppliers:
            if s.name.startswith('А'):
                s_details = list(filter(lambda i: i[2] == s.name, many_to_many))
```

```
                s_detail_names = [name for name, _, _ in s_details]
                if s_detail_names:  # Добавляем только если есть детали
                    res[s.name] = s_detail_names
        return res

def main():
    """Основная функция"""
    data_service = DataService()
    query_service = QueryService()

    # Получение данных
    suppliers, details, detail_suppliers = data_service.create_test_data()
    one_to_many = data_service.create_one_to_many(suppliers, details)
    many_to_many = data_service.create_many_to_many(suppliers, details, detail_suppliers)

    # Выполнение запросов
    print('Задание Д1:')
    result1 = query_service.query1(one_to_many)
    print(result1)

    print('\nЗадание Д2:')
    result2 = query_service.query2(one_to_many, suppliers)
    print(result2)

    print('\nЗадание Д3:')
    result3 = query_service.query3(many_to_many, suppliers)
    print(result3)

    return result1, result2, result3

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

test_rk1.py

```
# test_rk1.py
import unittest
from rk1_refactored import DataService, QueryService, Detail, Supplier, DetailSupplier

class TestRK1(unittest.TestCase):
    """Тесты для рубежного контроля №1"""

    def setUp(self):
        """Подготовка тестовых данных"""
        self.data_service = DataService()
        self.query_service = QueryService()
        self.suppliers, self.details, self.detail_suppliers =
self.data_service.create_test_data()
        self.one_to_many = self.data_service.create_one_to_many(self.suppliers, self.details)
        self.many_to_many = self.data_service.create_many_to_many(
            self.suppliers, self.details, self.detail_suppliers
        )
```

```
def test_query1_details_ending_with_ov(self):
    """Тест 1: Проверка запроса деталей с названием на 'ов'"""
    # Act
    result = self.query_service.query1(self.one_to_many)

    # Assert
    # Проверяем, что все найденные детали заканчиваются на 'ов'
    for detail in result:
        self.assertTrue(detail[0].endswith('ов'),
                        f"Деталь '{detail[0]}' не заканчивается на 'ов'")

    # Проверяем количество найденных деталей
    self.assertEqual(len(result), 4)

    # Проверяем конкретные детали
    detail_names = [detail[0] for detail in result]
    expected_names = ['Болтов', 'Винтов', 'Шурупов', 'Резьбов']
    for name in expected_names:
        self.assertIn(name, detail_names)

def test_query2_average_cost_per_supplier(self):
    """Тест 2: Проверка расчета средней стоимости деталей по поставщикам"""
    # Act
    result = self.query_service.query2(self.one_to_many, self.suppliers)

    # Assert
    # Проверяем количество поставщиков
    self.assertEqual(len(result), 4)

    # Проверяем правильность расчетов
    expected_results = [
        ('Авангард', 1350.0),
        ('Бетакон', 2000.0),
        ('Агропром', 1800.0),
        ('Металлинвест', 2200.0)
    ]

    for expected in expected_results:
        found = False
        for actual in result:
            if actual[0] == expected[0]:
                self.assertAlmostEqual(actual[1], expected[1], places=2,
                                       msg=f"Неверная средняя стоимость для {expected[0]}")
                found = True
                break
        self.assertTrue(found, f"Поставщик {expected[0]} не найден в результатах")

    # Проверяем сортировку по возрастанию средней стоимости
    for i in range(len(result) - 1):
        self.assertLessEqual(result[i][1], result[i+1][1],
                             "Результаты не отсортированы по возрастанию")

def test_query3_suppliers_starting_with_a(self):
```

```
        """Тест 3: Проверка запроса поставщиков на 'A' и их деталей"""
        # Act
        result = self.query_service.query3(self.many_to_many, self.suppliers)

        # Assert
        # Проверяем количество поставщиков на 'A'
        self.assertEqual(len(result), 2)

        # Проверяем конкретных поставщиков
        self.assertIn('Авангард', result)
        self.assertIn('Агропром', result)

        # Проверяем детали поставщика 'Авангард'
        self.assertEqual(len(result['Авангард']), 2)
        self.assertIn('Болтов', result['Авангард'])
        self.assertIn('Шурупов', result['Авангард'])

        # Проверяем детали поставщика 'Агропром'
        self.assertEqual(len(result['Агропром']), 1)
        self.assertIn('Гайков', result['Агропром'])

        # Проверяем, что нет других поставщиков
        for supplier_name in result.keys():
            self.assertTrue(supplier_name.startswith('A'),
                            f"Поставщик '{supplier_name}' не начинается с 'A'")

class TestDataService(unittest.TestCase):
    """Тесты для сервиса данных"""

    def test_create_one_to_many(self):
        """Тест создания связи один-ко-многим"""
        # Arrange
        suppliers = [Supplier(1, 'TestSupplier')]
        details = [
            Detail(1, 'TestDetail1', 100, 1),
            Detail(2, 'TestDetail2', 200, 1),
            Detail(3, 'TestDetail3', 300, 2),  # Другая связь
        ]

        # Act
        result = DataService.create_one_to_many(suppliers, details)

        # Assert
        self.assertEqual(len(result), 2)
        self.assertEqual(result[0][0], 'TestDetail1')
        self.assertEqual(result[0][1], 100)
        self.assertEqual(result[0][2], 'TestSupplier')

if __name__ == '__main__':
    # Запуск тестов
    unittest.main(verbosity=2)
```

run_tests.py

```
# run_tests.py
import unittest
import sys

if __name__ == '__main__':
    # Загрузка тестов из модуля test_rk1
    test_loader = unittest.TestLoader()
    test_suite = test_loader.discover('.', pattern='test_*.py')

    # Запуск тестов
    test_runner = unittest.TextTestRunner(verbosity=2)
    result = test_runner.run(test_suite)

    # Возврат кода выхода в зависимости от результата тестов
    sys.exit(0 if result.wasSuccessful() else 1)
```

Результат выполнения

```
zhuwen@boxj:~/Zhuwen-CS-Labs-2025/RK2$ python3 run_tests.py
test_create_one_to_many (test_rk1.TestDataService.test_create_one_to_many)
Тест создания связи один-ко-многим ... ok
test_query1_details_ending_with_ov (test_rk1.TestRK1.test_query1_details_ending_with_ov)
Тест 1: Проверка запроса деталей с названием на 'ов' ... ok
test_query2_average_cost_per_supplier (test_rk1.TestRK1.test_query2_average_cost_per_supplier)
Тест 2: Проверка расчета средней стоимости деталей по поставщикам ... ok
test_query3_suppliers_starting_with_a (test_rk1.TestRK1.test_query3_suppliers_starting_with_a)
Тест 3: Проверка запроса поставщиков на 'А' и их деталей ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
```