**Московский государственный технический
университет им. Н. Э. Баумана**

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

Курс «Парадигмы и конструкции языков программирования»
Отчет по рубежному контролю №2
«**Вариант А, 19**»

Выполнил:
Студент группы ИУ5-31Б
Цориев Никита

Проверил:
Гапанюк Ю. Е.

2025 г.

Листинг программы:

main_demo.py

```python
from refactored_manufacturing import (
    get_sample_data,
    ManufacturingService,
    task1_functional,
    task2_functional,
    task3_functional
)


def demonstrate_refactored_code():
    print("ДЕМОНСТРАЦИЯ РЕФАКТОРИНГА")
    print("=" * 60)

    manufacturers, details, manufacturer_details = get_sample_data()

    service = ManufacturingService(manufacturers, details, manufacturer_details)

    # Демонстрируем каждый метод
    print("\n1. Детали с производителями:")
    print("-" * 40)
    details_by_manufacturer = service.get_details_by_manufacturer()
    for i, (manufacturer, detail, price) in enumerate(details_by_manufacturer[:3], 1):
        print(f"{i}. {manufacturer} -> {detail} ({price} руб.)")
    print(f"... и еще {len(details_by_manufacturer) - 3} записей")

    print("\n2. Суммарная стоимость по производителям:")
    print("-" * 40)
    total_prices = service.get_total_price_by_manufacturer()
    for manufacturer, total in total_prices:
        print(f"{manufacturer}: {total:.2f} руб.")

    print("\n3. Производители с 'отдел' в названии:")
    print("-" * 40)
    department_manufacturers = service.get_department_manufacturers_with_details()
    for manufacturer, details_list in department_manufacturers.items():
        print(f"\n{manufacturer}:")
        for detail, price in details_list[:2]:
            print(f"  - {detail} ({price} руб.)")
        if len(details_list) > 2:
            print(f"  ... и еще {len(details_list) - 2} деталей")


def demonstrate_original_functionality():
    print("\n\n" + "=" * 60)
    print("ДЕМОНСТРАЦИЯ ИСХОДНОЙ ФУНКЦИОНАЛЬНОСТИ")
    print("=" * 60)

    task1_functional()
    task2_functional()
    task3_functional()
```

```python
if __name__ == "__main__":

    demonstrate_refactored_code()

    demonstrate_original_functionality()
```

test_manufacturing.py

```python
import unittest
from refactored_manufacturing import (
    Manufacturer,
    Detail,
    ManufacturerDetail,
    ManufacturingService,
    get_sample_data
)


class TestManufacturingService(unittest.TestCase):
    """Модульные тесты для ManufacturingService"""

    def setUp(self):
        """Настройка тестовых данных перед каждым тестом"""
        self.manufacturers, self.details, self.manufacturer_details = get_sample_data()
        self.service = ManufacturingService(
            self.manufacturers,
            self.details,
            self.manufacturer_details
        )

    def test_get_details_by_manufacturer(self):
        """Тест 1: Проверка получения деталей с производителями"""

        result = self.service.get_details_by_manufacturer()

        self.assertEqual(len(result), 10)

        first_record = result[0]
        self.assertEqual(first_record[0], "Основной производственный отдел")
        self.assertEqual(first_record[1], "Подшипник 6000")
        self.assertEqual(first_record[2], 45.00)

        manufacturer_ids = []
        for detail in self.details:
            manufacturer_ids.append(detail.manufacturer_id)

        self.assertEqual(result[0][0], "Основной производственный отдел")
        self.assertEqual(result[1][0], "Основной производственный отдел")

        print("✓ Тест 1 пройден: get_details_by_manufacturer работает корректно")

    def test_get_total_price_by_manufacturer(self):
        """Тест 2: Проверка расчета суммарной стоимости по производителям"""
        result = self.service.get_total_price_by_manufacturer()

        self.assertEqual(len(result), 5)
```

```
        for manufacturer_name, total_price in result:
            if manufacturer_name == "Отдел металлообработки":
                self.assertAlmostEqual(total_price, 29.00, places=2)
                break

        prices = [price for _, price in result]
        self.assertEqual(prices, sorted(prices, reverse=True))

        for manufacturer_name, total_price in result:
            self.assertGreaterEqual(total_price, 0.0)

        print("✓ Тест 2 пройден: get_total_price_by_manufacturer работает корректно")

    def test_get_department_manufacturers_with_details(self):
        """Тест 3: Проверка фильтрации производителей с 'отдел' в названии"""
        result = self.service.get_department_manufacturers_with_details()

        expected_department_manufacturers = [
            "Основной производственный отдел",
            "Отдел металлообработки",
            "Электротехнический отдел",
            "Отдел крепежных изделий"
        ]

        self.assertEqual(len(result), 4)

        for manufacturer_name in expected_department_manufacturers:
            self.assertIn(manufacturer_name, result)

        manufacturer_name = "Отдел металлообработки"
        details_list = result[manufacturer_name]

        self.assertEqual(len(details_list), 3)

        detail_names = [detail_name for detail_name, _ in details_list]
        expected_detail_names = ["Болт М8", "Гайка М8", "Шайба 8мм"]

        for expected_name in expected_detail_names:
            self.assertIn(expected_name, detail_names)

        self.assertNotIn("Производитель электронных компонентов", result)

        print("✓ Тест 3 пройден: get_department_manufacturers_with_details работает корректно")

    def test_empty_data(self):
        """Тест с пустыми данными"""
        empty_service = ManufacturingService([], [], [])
```

```
        result1 = empty_service.get_details_by_manufacturer()
        result2 = empty_service.get_total_price_by_manufacturer()
        result3 = empty_service.get_department_manufacturers_with_details()

        self.assertEqual(result1, [])
        self.assertEqual(result2, [])
        self.assertEqual(result3, {})

        print("✓ Тест с пустыми данными пройден")


class TestDataStructures(unittest.TestCase):
    """Тесты для структур данных"""

    def test_manufacturer_creation(self):
        """Тест создания объекта Manufacturer"""
        manufacturer = Manufacturer(1, "Тестовый производитель")

        self.assertEqual(manufacturer.manufacturer_id, 1)
        self.assertEqual(manufacturer.manufacturer_name, "Тестовый производитель")
        self.assertEqual(str(manufacturer),
                         "Manufacturer(manufacturer_id=1, manufacturer_name='Тестовый производитель')")

    def test_detail_creation(self):
        """Тест создания объекта Detail"""
        detail = Detail(1, "Тестовая деталь", 100.50, 1)

        self.assertEqual(detail.detail_id, 1)
        self.assertEqual(detail.detail_name, "Тестовая деталь")
        self.assertEqual(detail.price, 100.50)
        self.assertEqual(detail.manufacturer_id, 1)


def run_tests():
    """Запуск всех тестов"""
    print("=" * 60)
    print("ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ")
    print("=" * 60)

    loader = unittest.TestLoader()
    suite = unittest.TestSuite()

    suite.addTests(loader.loadTestsFromTestCase(TestManufacturingService))
    suite.addTests(loader.loadTestsFromTestCase(TestDataStructures))

    runner = unittest.TextTestRunner(verbosity=2)
    result = runner.run(suite)
```

```
    print("\n" + "=" * 60)
    print("ИТОГИ ТЕСТИРОВАНИЯ")
    print("=" * 60)
    print(f"Всего тестов: {result.testsRun}")
    print(f"Успешно: {result.testsRun - len(result.failures) - len(result.errors)}")
    print(f"Провалено: {len(result.failures)}")
    print(f"Ошибок: {len(result.errors)}")

    if result.wasSuccessful():
        print("\n✓ ВСЕ ТЕСТЫ ПРОЙДЕНЫ УСПЕШНО!")
    else:
        print("\n✗ ЕСТЬ ПРОБЛЕМЫ С ТЕСТАМИ")

    return result.wasSuccessful()


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

run_all.py

```python
if __name__ == "__main__":
    print("РУБЕЖНЫЙ КОНТРОЛЬ: РЕФАКТОРИНГ И ТЕСТИРОВАНИЕ")
    print("=" * 60)

    from main_demo import (
        demonstrate_refactored_code,
        demonstrate_original_functionality
    )

    demonstrate_refactored_code()
    demonstrate_original_functionality()

    print("\n\n" + "=" * 60)
    print("ДЛЯ ЗАПУСКА ТЕСТОВ:")
    print("=" * 60)
    print("Выполните одну из следующих команд в терминале:")
    print("\n1. Запуск тестов через unittest:")
    print("   python -m unittest test_manufacturing.py")
    print("\n2. Запуск тестового файла напрямую:")
    print("   python test_manufacturing.py")
    print("\n3. Запуск с более подробным выводом:")
    print("   python -m unittest test_manufacturing.TestManufacturingService -v")
```

refactored_manufacturing.py

```python
from dataclasses import dataclass
from typing import List, Dict, Tuple
from collections import defaultdict

@dataclass
class Manufacturer:
    """Класс для представления производителя"""
    manufacturer_id: int
    manufacturer_name: str

@dataclass
class Detail:
    """Класс для представления детали"""
    detail_id: int
    detail_name: str
    price: float
    manufacturer_id: int

@dataclass
class ManufacturerDetail:
    """Класс для связи производителя и детали"""
    manufacturer_id: int
    detail_id: int


class ManufacturingService:
    """Сервис для работы с данными о производителях и деталях"""

    def __init__(self, manufacturers: List[Manufacturer],
                 details: List[Detail],
                 manufacturer_details: List[ManufacturerDetail]):
        self.manufacturers = manufacturers
        self.details = details
        self.manufacturer_details = manufacturer_details

    def get_manufacturers_dict(self) -> Dict[int, Manufacturer]:
        """Создает словарь производителей по ID"""
        return {m.manufacturer_id: m for m in self.manufacturers}

    def get_details_dict(self) -> Dict[int, Detail]:
        """Создает словарь деталей по ID"""
        return {d.detail_id: d for d in self.details}

    def get_manufacturer_to_details(self) -> Dict[int, List[int]]:
        """Создает словарь соответствия производителя и его деталей"""
        manufacturer_to_details = defaultdict(list)
```

```python
    for md in self.manufacturer_details:
        manufacturer_to_details[md.manufacturer_id].append(md.detail_id)
    return manufacturer_to_details

def get_details_by_manufacturer(self) -> List[Tuple[str, str, float]]:
    """Запрос 1: Получить детали с их производителями"""
    manufacturer_dict = self.get_manufacturers_dict()
    result = []

    for detail in sorted(self.details, key=lambda x: (x.manufacturer_id, x.detail_id)):
        manufacturer = manufacturer_dict.get(detail.manufacturer_id)
        if manufacturer:
            result.append((
                manufacturer.manufacturer_name,
                detail.detail_name,
                detail.price
            ))
    return result

def get_total_price_by_manufacturer(self) -> List[Tuple[str, float]]:
    """Запрос 2: Получить суммарную стоимость деталей по производителям"""
    price_totals = defaultdict(float)

    for detail in self.details:
        price_totals[detail.manufacturer_id] += detail.price

    manufacturer_dict = self.get_manufacturers_dict()
    result = []

    for manufacturer in self.manufacturers:
        total_price = price_totals.get(manufacturer.manufacturer_id, 0.0)
        result.append((manufacturer.manufacturer_name, total_price))

    # Сортировка по убыванию суммарной стоимости
    return sorted(result, key=lambda x: x[1], reverse=True)

def get_department_manufacturers_with_details(self) -> Dict[str, List[Tuple[str, float]]]:
    """Запрос 3: Получить производителей с 'отдел' в названии и их детали"""
    manufacturer_dict = self.get_manufacturers_dict()
    detail_dict = self.get_details_dict()
    manufacturer_to_details = self.get_manufacturer_to_details()

    result = {}
```

```
        for manufacturer in self.manufacturers:
            if "отдел" in manufacturer.manufacturer_name.lower():
                detail_ids = manufacturer_to_details.get(manufacturer.manufacturer_id, [])
                details_list = []

to add a breakpoint  for detail_id in detail_ids:
                    if detail_id in detail_dict:
                        detail = detail_dict[detail_id]
                        details_list.append((detail.detail_name, detail.price))

                result[manufacturer.manufacturer_name] = details_list

        return result


# Данные для работы
def get_sample_data() -> Tuple[List[Manufacturer], List[Detail], List[ManufacturerDetail]]:
    """Возвращает тестовые данные"""
    manufacturers = [
        Manufacturer(1, "Основной производственный отдел"),
        Manufacturer(2, "Отдел металлообработки"),
        Manufacturer(3, "Электротехнический отдел"),
        Manufacturer(4, "Производитель электронных компонентов"),
        Manufacturer(5, "Отдел крепежных изделий")
    ]

    details = [
        Detail(1, "Болт М8", 15.50, 2),
        Detail(2, "Гайка М8", 8.30, 2),
        Detail(3, "Шайба 8мм", 5.20, 2),
        Detail(4, "Микроконтроллер ATmega328", 250.00, 4),
        Detail(5, "Резистор 100 Ом", 2.50, 4),
        Detail(6, "Конденсатор 10мкФ", 3.80, 4),
        Detail(7, "Винт саморез", 12.00, 5),
        Detail(8, "Дюбель пластиковый", 7.50, 5),
        Detail(9, "Подшипник 6000", 45.00, 1),
        Detail(10, "Шестерня модуль 1", 120.00, 1)
    ]
```

```
    manufacturer_details = [
        ManufacturerDetail(1, 9), ManufacturerDetail(1, 10),
        ManufacturerDetail(2, 1), ManufacturerDetail(2, 2), ManufacturerDetail(2, 3),
        ManufacturerDetail(3, 5), ManufacturerDetail(3, 6),
        ManufacturerDetail(4, 4), ManufacturerDetail(4, 5), ManufacturerDetail(4, 6),
        ManufacturerDetail(5, 7), ManufacturerDetail(5, 8)
    ]

    return manufacturers, details, manufacturer_details


def task1_functional():
    manufacturers, details, manufacturer_details = get_sample_data()
    service = ManufacturingService(manufacturers, details, manufacturer_details)
    result = service.get_details_by_manufacturer()

    print("=== ЗАПРОС 1 (функциональный): Детали и производители ===")
    for manufacturer_name, detail_name, price in result:
        print(f"Производитель: {manufacturer_name} -> "
              f"Деталь: {detail_name}, Цена: {price} руб.")
    print()

def task2_functional():
    manufacturers, details, manufacturer_details = get_sample_data()
    service = ManufacturingService(manufacturers, details, manufacturer_details)
    result = service.get_total_price_by_manufacturer()

    print("=== ЗАПРОС 2 (функциональный): Суммарная стоимость ===")
    for manufacturer_name, total_price in result:
        print(f"Производитель: {manufacturer_name}, Суммарная стоимость: {total_price:.2f} руб.")
    print()

def task3_functional():
    manufacturers, details, manufacturer_details = get_sample_data()
    service = ManufacturingService(manufacturers, details, manufacturer_details)
    result = service.get_department_manufacturers_with_details()

    print("=== ЗАПРОС 3 (функциональный): Производители с 'отдел' в названии ===")
    for manufacturer_name, details_list in result.items():
        print(f"\nПроизводитель: {manufacturer_name}")
        if details_list:
            for detail_name, price in details_list:
                print(f"  - Деталь: {detail_name}, Цена: {price} руб.")
        else:
```

```
            print("  - Нет деталей")
    print()


if __name__ == "__main__":
    print("ФУНКЦИОНАЛЬНАЯ РЕАЛИЗАЦИЯ:")
    print("=" * 50)
    task1_functional()
    task2_functional()
    task3_functional()
```

Результат выполнения:

```
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$ python3 run_all.py
РУБЕЖНЫЙ КОНТРОЛЬ: РЕФАКТОРИНГ И ТЕСТИРОВАНИЕ
============================================================
ДЕМОНСТРАЦИЯ РЕФАКТОРИНГА
============================================================

1. Детали с производителями:
--------------------------------------
1. Основной производственный отдел -> Подшипник 6000 (45.0 руб.)
2. Основной производственный отдел -> Шестерня модуль 1 (120.0 руб.)
3. Отдел металлообработки -> Болт М8 (15.5 руб.)
... и еще 7 записей

2. Суммарная стоимость по производителям:
--------------------------------------
Производитель электронных компонентов: 256.30 руб.
Основной производственный отдел: 165.00 руб.
Отдел металлообработки: 29.00 руб.
Отдел крепежных изделий: 19.50 руб.
Электротехнический отдел: 0.00 руб.

3. Производители с 'отдел' в названии:
--------------------------------------

Основной производственный отдел:
  - Подшипник 6000 (45.0 руб.)
  - Шестерня модуль 1 (120.0 руб.)

Отдел металлообработки:
  - Болт М8 (15.5 руб.)
  - Гайка М8 (8.3 руб.)
  ... и еще 1 деталей

Электротехнический отдел:
  - Резистор 100 Ом (2.5 руб.)
  - Конденсатор 10мкФ (3.8 руб.)

Отдел крепежных изделий:
  - Винт саморез (12.0 руб.)
  - Дюбель пластиковый (7.5 руб.)
```

```
============================================================
ДЕМОНСТРАЦИЯ ИСХОДНОЙ ФУНКЦИОНАЛЬНОСТИ
============================================================
=== ЗАПРОС 1 (функциональный): Детали и производители ===
Производитель: Основной производственный отдел -> Деталь: Подшипник 6000, Цена: 45.0 руб.
Производитель: Основной производственный отдел -> Деталь: Шестерня модуль 1, Цена: 120.0 руб.
Производитель: Отдел металлообработки -> Деталь: Болт М8, Цена: 15.5 руб.
Производитель: Отдел металлообработки -> Деталь: Гайка М8, Цена: 8.3 руб.
Производитель: Отдел металлообработки -> Деталь: Шайба 8мм, Цена: 5.2 руб.
Производитель: Производитель электронных компонентов -> Деталь: Микроконтроллер ATmega328, Цена: 250.0 руб.
Производитель: Производитель электронных компонентов -> Деталь: Резистор 100 Ом, Цена: 2.5 руб.
Производитель: Производитель электронных компонентов -> Деталь: Конденсатор 10мкФ, Цена: 3.8 руб.
Производитель: Отдел крепежных изделий -> Деталь: Винт саморез, Цена: 12.0 руб.
Производитель: Отдел крепежных изделий -> Деталь: Дюбель пластиковый, Цена: 7.5 руб.

=== ЗАПРОС 2 (функциональный): Суммарная стоимость ===
Производитель: Производитель электронных компонентов, Суммарная стоимость: 256.30 руб.
Производитель: Основной производственный отдел, Суммарная стоимость: 165.00 руб.
Производитель: Отдел металлообработки, Суммарная стоимость: 29.00 руб.
Производитель: Отдел крепежных изделий, Суммарная стоимость: 19.50 руб.
Производитель: Электротехнический отдел, Суммарная стоимость: 0.00 руб.

=== ЗАПРОС 3 (функциональный): Производители с 'отдел' в названии ===

Производитель: Основной производственный отдел
  - Деталь: Подшипник 6000, Цена: 45.0 руб.
  - Деталь: Шестерня модуль 1, Цена: 120.0 руб.

Производитель: Отдел металлообработки
  - Деталь: Болт М8, Цена: 15.5 руб.
  - Деталь: Гайка М8, Цена: 8.3 руб.
  - Деталь: Шайба 8мм, Цена: 5.2 руб.

Производитель: Электротехнический отдел
  - Деталь: Резистор 100 Ом, Цена: 2.5 руб.
  - Деталь: Конденсатор 10мкФ, Цена: 3.8 руб.

Производитель: Отдел крепежных изделий
  - Деталь: Винт саморез, Цена: 12.0 руб.
  - Деталь: Дюбель пластиковый, Цена: 7.5 руб.
```

```
============================================================
ДЛЯ ЗАПУСКА ТЕСТОВ:
============================================================
Выполните одну из следующих команд в терминале:

1. Запуск тестов через unittest:
   python -m unittest test_manufacturing.py

2. Запуск тестового файла напрямую:
   python test_manufacturing.py

3. Запуск с более подробным выводом:
   python -m unittest test_manufacturing.TestManufacturingService -v
```

```
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$ python3 -m unittest test_manufacturing.py
..✓ Тест с пустыми данными пройден
.✓ Тест 3 пройден: get_department_manufacturers_with_details работает корректно
.✓ Тест 1 пройден: get_details_by_manufacturer работает корректно
.✓ Тест 2 пройден: get_total_price_by_manufacturer работает корректно
.
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$
```

```
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$ python3 test_manufacturing.py
============================================================
ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ
============================================================
test_empty_data (__main__.TestManufacturingService.test_empty_data)
Тест с пустыми данными ... ✓ Тест с пустыми данными пройден
ok
test_get_department_manufacturers_with_details (__main__.TestManufacturingService.test_get_department_manufacturers_with_details)
Тест 3: Проверка фильтрации производителей с 'отдел' в названии ... ✓ Тест 3 пройден: get_department_manufacturers_with_details работает коррек
тно
ok
test_get_details_by_manufacturer (__main__.TestManufacturingService.test_get_details_by_manufacturer)
Тест 1: Проверка получения деталей с производителями ... ✓ Тест 1 пройден: get_details_by_manufacturer работает корректно
ok
test_get_total_price_by_manufacturer (__main__.TestManufacturingService.test_get_total_price_by_manufacturer)
Тест 2: Проверка расчета суммарной стоимости по производителям ... ✓ Тест 2 пройден: get_total_price_by_manufacturer работает корректно
ok
test_detail_creation (__main__.TestDataStructures.test_detail_creation)
Тест создания объекта Detail ... ok
test_manufacturer_creation (__main__.TestDataStructures.test_manufacturer_creation)
Тест создания объекта Manufacturer ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK

============================================================
ИТОГИ ТЕСТИРОВАНИЯ
============================================================
Всего тестов: 6
Успешно: 6
Провалено: 0
Ошибок: 0

✓ ВСЕ ТЕСТЫ ПРОЙДЕНЫ УСПЕШНО!
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$
```

```
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$ python3 -m unittest test_manufacturing.TestManufacturingService -v
test_empty_data (test_manufacturing.TestManufacturingService.test_empty_data)
Тест с пустыми данными ... ✓ Тест с пустыми данными пройден
ok
test_get_department_manufacturers_with_details (test_manufacturing.TestManufacturingService.test_get_department_manufacturers_with_details)
Тест 3: Проверка фильтрации производителей с 'отдел' в названии ... ✓ Тест 3 пройден: get_department_manufacturers_with_details работает коррек
тно
ok
test_get_details_by_manufacturer (test_manufacturing.TestManufacturingService.test_get_details_by_manufacturer)
Тест 1: Проверка получения деталей с производителями ... ✓ Тест 1 пройден: get_details_by_manufacturer работает корректно
ok
test_get_total_price_by_manufacturer (test_manufacturing.TestManufacturingService.test_get_total_price_by_manufacturer)
Тест 2: Проверка расчета суммарной стоимости по производителям ... ✓ Тест 2 пройден: get_total_price_by_manufacturer работает корректно
ok

----------------------------------------------------------------------
Ran 4 tests in 0.002s

OK
tsoriev_nv@LAPTOP-SFRKJ040:~/rkPLP/rkPLP/rk2$
```