Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНЖЕНЕРНЫЙ БИЗНЕС И МЕНЕДЖМЕНТ»
КАФЕДРА «ПРОМЫШЛЕННАЯ ЛОГИСТИКА»

ОТЧЕТ
Предмет: Парадигмы и конструкции языков программирования
Рубежный контроль №2

Студент: Сигорский Александр Евгеньевич

Группа: ИБМ3-34Б

Преподаватель: Ю.Е. Гапанюк

Кафедра: ИБМ-3

Москва 2025

Задание:

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 применение TDD-фреймворка(3 теста)

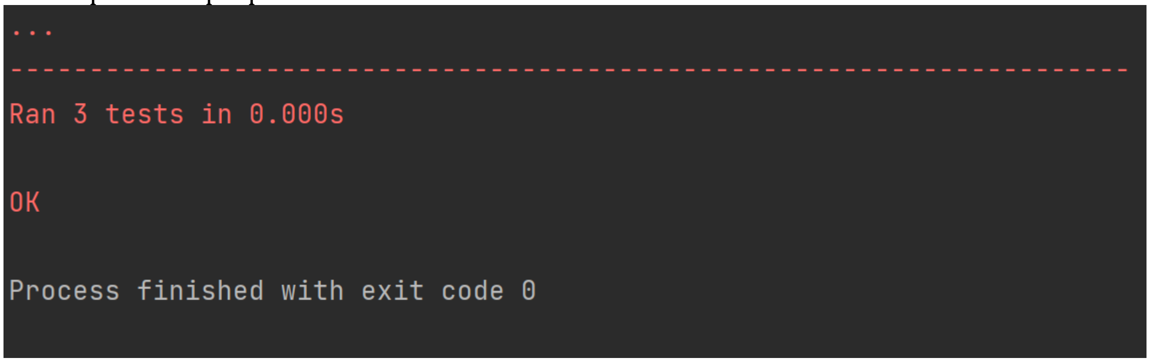
Код программы:

```
from dataclasses import dataclass

from typing import List, Tuple
import unittest


@dataclass
class Supplier:
    id: int
    name: str


@dataclass
class Part:
    id: int
    name: str
    cost: float
    supplier_id: int


@dataclass
class SupplierPart:
    supplier_id: int
    part_id: int


SUPPLIERS: List[Supplier] = [
    Supplier(1, "AutoParts LLC"),
    Supplier(2, "Ivanov IE - metals department"),
    Supplier(3, "Mechanics CJSC"),
    Supplier(4, "Industrial LLC"),
    Supplier(5, "Trade Company electronics department"),
]

PARTS: List[Part] = [
    Part(1, "Crankshaft pulley", 1250.50, 1),
    Part(2, "Piston", 890.00, 1),
    Part(3, "Brake pad", 340.75, 1),
    Part(4, "Suspension lever", 1560.00, 2),
    Part(5, "Body panel", 2100.25, 2),
    Part(6, "Shock absorber", 980.00, 3),
    Part(7, "Nut M12", 15.50, 3),
    Part(8, "Oil filter", 250.00, 3),
    Part(9, "Spark plug", 120.00, 5),
    Part(10, "Oxygen sensor", 1850.00, 5),
]

SUPPLIER_PARTS: List[SupplierPart] = [
    SupplierPart(1, 1),
    SupplierPart(1, 2),
    SupplierPart(1, 3),
    SupplierPart(2, 4),
    SupplierPart(2, 5),
    SupplierPart(3, 6),
    SupplierPart(3, 7),
    SupplierPart(3, 8),
    SupplierPart(5, 9),
    SupplierPart(5, 10),
    SupplierPart(2, 6),
    SupplierPart(3, 1),
    SupplierPart(4, 2),
]


def query_one_to_many(
    suppliers: List[Supplier], parts: List[Part]
) -> List[Tuple[Part, Supplier]]:
    """Return one-to-many pairs sorted by supplier id then part name."""
    pairs = [
        (part, supplier)
        for supplier in suppliers
        for part in parts
        if part.supplier_id == supplier.id
    ]
    return sorted(pairs, key=lambda item: (item[1].id, item[0].name))


def query_supplier_totals(
    suppliers: List[Supplier], parts: List[Part]
) -> List[Tuple[Supplier, float]]:
    """Return suppliers with total part cost sorted by total ascending."""
    totals = {
        supplier.id: sum(part.cost for part in parts if part.supplier_id == supplier.id)
        for supplier in suppliers
    }
    result = [(supplier, totals[supplier.id]) for supplier in suppliers if totals[supplier.id] > 0]
    return sorted(result, key=lambda item: item[1])
```

```
def query_departments(
    suppliers: List[Supplier],
    supplier_parts: List[SupplierPart],
    parts: List[Part],
) -> List[Tuple[Supplier, List[Part]]]:
    """Return suppliers with 'department' in name and their parts via many-to-many."""
    result: List[Tuple[Supplier, List[Part]]] = []
    department_suppliers = [s for s in suppliers if "department" in s.name.lower()]

    for supplier in sorted(department_suppliers, key=lambda s: s.id):
        related_part_ids = [link.part_id for link in supplier_parts if link.supplier_id == supplier.id]
        related_parts = [part for part in parts if part.id in related_part_ids]
        result.append((supplier, related_parts))
    return result


class TestQueries(unittest.TestCase):
    def setUp(self) -> None:
        self.suppliers = SUPPLIERS
        self.parts = PARTS
        self.supplier_parts = SUPPLIER_PARTS

    def test_query_one_to_many_returns_sorted_pairs(self) -> None:
        pairs = query_one_to_many(self.suppliers, self.parts)
        self.assertEqual(len(pairs), 10)
        first_part, first_supplier = pairs[0]
        self.assertEqual(first_part.name, "Brake pad")
        self.assertEqual(first_supplier.name, "AutoParts LLC")

    def test_query_supplier_totals_orders_by_cost(self) -> None:
        totals = query_supplier_totals(self.suppliers, self.parts)
        names_with_totals = [(supplier.name, total) for supplier, total in totals]
        expected = [
            ("Mechanics CJSC", 1245.50),
            ("Trade Company electronics department", 1970.00),
            ("AutoParts LLC", 2481.25),
            ("Ivanov IE - metals department", 3660.25),
        ]
        self.assertEqual(names_with_totals, expected)

    def test_query_departments_returns_related_parts(self) -> None:
        departments = query_departments(self.suppliers, self.supplier_parts, self.parts)
        self.assertEqual(len(departments), 2)
        supplier2, parts2 = departments[0]
        supplier5, parts5 = departments[1]

        self.assertEqual(supplier2.name, "Ivanov IE - metals department")
        self.assertEqual({p.id for p in parts2}, {4, 5, 6})

        self.assertEqual(supplier5.name, "Trade Company electronics department")
        self.assertEqual({p.id for p in parts5}, {9, 10})
```

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

Скриншот работы программы:



```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK

Process finished with exit code 0
```