import java.io.\*;

import java.util.\*;

import java.util.concurrent.\*;

class Location {

private int aisle, shelf, bin;

public Location(int aisle, int shelf, int bin) {

this.aisle = aisle;

this.shelf = shelf;

this.bin = bin;

}

@Override

public String toString() {

return "Aisle: " + aisle + ", Shelf: " + shelf + ", Bin: " + bin;

}

}

class Product {

private String productID, name;

private int quantity;

private Location location;

public Product(String productID, String name, int quantity, Location location) {

this.productID = productID;

this.name = name;

this.quantity = quantity;

this.location = location;

}

public String getProductID() { return productID; }

public int getQuantity() { return quantity; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public Location getLocation() { return location; }

@Override

public String toString() {

return "Product ID: " + productID + ", Name: " + name + ", Quantity: " + quantity + ", Location: " + location;

}

}

class OutOfStockException extends Exception {

public OutOfStockException(String message) {

super(message);

}

}

class Order implements Comparable<Order> {

enum Priority { STANDARD, EXPEDITED }

private String orderID;

private List<String> productIDs;

private Priority priority;

public Order(String orderID, List<String> productIDs, Priority priority) {

this.orderID = orderID;

this.productIDs = productIDs;

this.priority = priority;

}

public List<String> getProductIDs() { return productIDs; }

public Priority getPriority() { return priority; }

@Override

public int compareTo(Order o) {

return this.priority.compareTo(o.priority);

}

}

class InventoryManager {

private ConcurrentHashMap<String, Product> products = new ConcurrentHashMap<>();

private PriorityQueue<Order> orderQueue = new PriorityQueue<>();

private ExecutorService executor = Executors.newFixedThreadPool(5);

public synchronized void addProduct(Product product) {

products.put(product.getProductID(), product);

}

public synchronized void placeOrder(Order order) {

orderQueue.add(order);

executor.submit(this::processOrders);

}

private void processOrders() {

while (!orderQueue.isEmpty()) {

Order order = orderQueue.poll();

for (String productID : order.getProductIDs()) {

try {

fulfillOrder(productID);

} catch (OutOfStockException e) {

System.out.println("Order failed: " + e.getMessage());

}

}

}

}

private synchronized void fulfillOrder(String productID) throws OutOfStockException {

Product product = products.get(productID);

if (product == null || product.getQuantity() <= 0) {

throw new OutOfStockException("Product " + productID + " is out of stock.");

}

product.setQuantity(product.getQuantity() - 1);

System.out.println("Order fulfilled for: " + productID);

}

}

public class WarehouseManagement {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

InventoryManager inventoryManager = new InventoryManager();

System.out.print("Enter number of products: ");

int numProducts = scanner.nextInt();

scanner.nextLine();

for (int i = 0; i < numProducts; i++) {

System.out.print("Enter Product ID: ");

String productID = scanner.nextLine();

System.out.print("Enter Product Name: ");

String name = scanner.nextLine();

System.out.print("Enter Quantity: ");

int quantity = scanner.nextInt();

System.out.print("Enter Aisle: ");

int aisle = scanner.nextInt();

System.out.print("Enter Shelf: ");

int shelf = scanner.nextInt();

System.out.print("Enter Bin: ");

int bin = scanner.nextInt();

scanner.nextLine();

inventoryManager.addProduct(new Product(productID, name, quantity, new Location(aisle, shelf, bin)));

}

System.out.print("Enter Order ID: ");

String orderID = scanner.nextLine();

System.out.print("Enter number of products in order: ");

int numOrderProducts = scanner.nextInt();

scanner.nextLine();

List<String> orderProducts = new ArrayList<>();

for (int i = 0; i < numOrderProducts; i++) {

System.out.print("Enter Product ID for Order: ");

orderProducts.add(scanner.nextLine());

}

System.out.print("Enter Order Priority (STANDARD/EXPEDITED): ");

Order.Priority priority = Order.Priority.valueOf(scanner.nextLine().toUpperCase());

Order order = new Order(orderID, orderProducts, priority);

inventoryManager.placeOrder(order);

scanner.close();

}

}