~\OneDrive - VietNam National University - HCM INTERNATIONAL UNIVERSITY\Desktop\DSA\DSA LAB NEW\Lab 2 Simple sorting\ITITSB22029_DoMinhDuy_Lab2\Problem 6\ECommerceOrderFulfillment.java

```java
import java.util.*;

public class ECommerceOrderFulfillment {

    static class Order {
        String orderId;
        int deadline; // in days
        int numItems;
        boolean fulfilled = false;
        String assignedWarehouse = "Unfulfilled";
        double completionTime; // The day when the order will be completed

        public Order(String orderId, int deadline, int numItems) {
            this.orderId = orderId;
            this.deadline = deadline;
            this.numItems = numItems;
        }
    }

    static class Warehouse {
        String warehouseId;
        double processingSpeed; // items per day
        double availableTime; // The day when the warehouse becomes available

        public Warehouse(String warehouseId, double processingSpeed) {
            this.warehouseId = warehouseId;
            this.processingSpeed = processingSpeed;
            this.availableTime = 0.0;
        }
    }

    public static void main(String[] args) {
        // Example input
        List<Order> orders = Arrays.asList(
                new Order("O1", 3, 50),
                new Order("O2", 1, 30),
                new Order("O3", 2, 40),
                new Order("O4", 2, 10),
                new Order("O5", 1, 20));

        List<Warehouse> warehouses = Arrays.asList(
                new Warehouse("W1", 20),
                new Warehouse("W2", 40),
                new Warehouse("W3", 15));

        // Fulfill the orders
        Map<String, List<Order>> schedule = fulfillOrders(orders, warehouses);

        // Output the schedule
        for (Warehouse warehouse : warehouses) {
            List<Order> assignedOrders = schedule.get(warehouse.warehouseId);
            if (assignedOrders != null && !assignedOrders.isEmpty()) {
                System.out.print("Warehouse " + warehouse.warehouseId + ": [");
                for (int i = 0; i < assignedOrders.size(); i++) {
                    Order order = assignedOrders.get(i);
                    System.out
                            .print("(" + "\"" + order.orderId + "\", " + order.deadline + ", "
+ order.numItems + ")");
                    if (i < assignedOrders.size() - 1) {
                        System.out.print(", ");
                    }
                }
                System.out.println("]");
            }
        }

        // Output unfulfilled orders
        List<Order> unfulfilledOrders = schedule.get("Unfulfilled");
        if (unfulfilledOrders != null && !unfulfilledOrders.isEmpty()) {
            System.out.print("Unfulfilled: [");
            for (int i = 0; i < unfulfilledOrders.size(); i++) {
                Order order = unfulfilledOrders.get(i);
                System.out.print("(" + "\"" + order.orderId + "\", " + order.deadline + ", " +
order.numItems + ")");
                if (i < unfulfilledOrders.size() - 1) {
                    System.out.print(", ");
                }
            }
            System.out.println("]");
        } else {
            System.out.println("Unfulfilled: []");
        }
    }

    public static Map<String, List<Order>> fulfillOrders(List<Order> orders, List<Warehouse>
warehouses) {
        // Sort orders based on earliest deadline first
        bubbleSortOrdersByDeadline(orders);

        // Sort warehouses by processing speed descending (highest speed first)
        bubbleSortWarehousesBySpeed(warehouses);

        // Map to hold the schedule
        Map<String, List<Order>> schedule = new LinkedHashMap<>();

        // Initialize schedule map with warehouse IDs
        for (Warehouse warehouse : warehouses) {
            schedule.put(warehouse.warehouseId, new ArrayList<>());
        }
        schedule.put("Unfulfilled", new ArrayList<>());

        for (Order order : orders) {
            Warehouse selectedWarehouse = null;
            double earliestCompletionTime = Double.MAX_VALUE;

            for (Warehouse warehouse : warehouses) {
                // Calculate processing time
                double processingTime = order.numItems / warehouse.processingSpeed;

                // Calculate completion time considering warehouse availability
                double completionTime = Math.max(warehouse.availableTime, 0) + processingTime;

                // Check if order can be fulfilled before the deadline
                if (completionTime <= order.deadline) {
                    // Select the warehouse with the earliest completion time
                    if (completionTime < earliestCompletionTime) {
                        earliestCompletionTime = completionTime;
                        selectedWarehouse = warehouse;
                    } else if (completionTime == earliestCompletionTime) {
                        // If tie, select the warehouse with higher processing speed
                        if (warehouse.processingSpeed > selectedWarehouse.processingSpeed) {
                            selectedWarehouse = warehouse;
                        }
                    }
                }
            }

            if (selectedWarehouse != null) {
                // Assign the order to the selected warehouse
                order.fulfilled = true;
                order.assignedWarehouse = selectedWarehouse.warehouseId;
                order.completionTime = earliestCompletionTime;

                // Update warehouse availability time
                selectedWarehouse.availableTime = earliestCompletionTime;

                // Add order to the warehouse's schedule
                schedule.get(selectedWarehouse.warehouseId).add(order);
            } else {
                // Order cannot be fulfilled
                schedule.get("Unfulfilled").add(order);
            }
        }

        return schedule;
    }

    public static void bubbleSortOrdersByDeadline(List<Order> orders) {
        int n = orders.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                // Swap if the current order has a later deadline than the next one
                if (orders.get(j).deadline > orders.get(j + 1).deadline) {
                    // Swap orders
                    Order temp = orders.get(j);
                    orders.set(j, orders.get(j + 1));
                    orders.set(j + 1, temp);
                }
            }
        }
    }

    public static void bubbleSortWarehousesBySpeed(List<Warehouse> warehouses) {
        int n = warehouses.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                // Swap if the current warehouse has a lower processing speed than the next one
                if (warehouses.get(j).processingSpeed < warehouses.get(j + 1).processingSpeed)
{
                    // Swap warehouses
                    Warehouse temp = warehouses.get(j);
                    warehouses.set(j, warehouses.get(j + 1));
                    warehouses.set(j + 1, temp);
                }
            }
        }
    }
}
```