Peer to Peer Parcel Delivery System (like Dunzo)

Description

Implement a Peer-to-Peer Delivery System that can be used to deliver a parcel from one customer to another. Below are the expected features of the system.
Problem Statement:

1. The system should be able to onboard new customers and drivers.
2. The list of items that can be delivered is preconfigured in the system and is fixed.
3. Customers should be able to place an order for the delivery of a parcel and also be able to cancel it.
4. One driver can pickup only one order at a time.
5. Orders should be auto-assigned to drivers based on availability. Even If no driver is available, the system should accept the order and assign it to a driver when the driver becomes free. The number of ongoing orders can exceed the number of drivers.
6. Once an order is assigned to a driver, the driver should be able to pick up the order and also mark the order as delivered after delivery.
7. The system should be able to show the status of orders and drivers.
8. Canceled orders shouldn't be assigned to the driver. If an assigned order gets canceled the driver shouldn't be able to pick up the order, the driver should be available for other orders.
9. Once a driver picks up an order the order cannot be canceled by the user nor system.
10. Assume driver is available 24*7. Ignore the travel time.
11. Ensure application is thread safe and all concurrency scenarios.

Bonus

- Notify the customer and drivers through email and phone for order updates. For this exercise, write a class that represents a vendor providing the email or SMS service and just print the log indicating that the vendor has processed the request
- Customers should be able to rate the driver after delivery.
- Dashboard to show top drivers based on different strategy no of orders, rating.
- If no driver picks up the order within 30 minutes of its creation, the order should be canceled. Regardless of whether an order has been assigned to a driver or not, if no driver picks it up within 30 minutes of order creation, the order should be canceled.

Guidelines:

- Time: 120 mins (Implementation).
- Write modular, clean and demo-able code (Test cases or runtime execution).
- **A driver program/main class/test case is needed to test out the code by the evaluator with multiple test cases**.
- **Use design patterns wherever applicable.**
- **Please handle concurrency wherever applicable.**

- **Evaluation criteria**: Demoable & functionally correct code, Code readability, Proper Entity modeling, Modularity & Extensibility, Separation of concerns, Abstractions, Exception Handling, Code comments.
- Code should handle edge cases properly and fail gracefully.
- You are not allowed to use any external databases like MySQL. Use only in-memory data structures.
- No need to create any UX or any HTTP API. It should be a standalone application.
- Usage of any AI powered tools such a chatGPT or github-copilot is **strictly** prohibited. You may use the internet to look up any syntactic references.
- You are free to use any popular programming language of your choice.
- **The bonus features are optional. Attempt them only after finishing the p0 features.**
- The problem may require you to make certain assumptions. Do make reasonable assumptions and convey them to the review panel.

Sample Test cases:
The input/output need not be in the same format this is for explaining the expected functionality
i: input
o: output

- onboard customer - id, name
- onboard driver - id, name
- create order - customer_id, item_id
- cancel order - order_id
- show order status - (order_id)
- show driver status - (driver_id)
- pick up order - (driver_id, order_id)
- complete order - (driver_id, order_id)