

Smart Warehouse

(Track stock, automate order)

Distributed Systems
Continuous Assessment

- Semester 2, 2022/23 -

Submitted by

DONGHYEOK LEE (21234175)

GitHub repo: <https://github.com/Donghyeoklee93/Project-Smart-Warehouse>
Video Presentation:

https://studentncirl-my.sharepoint.com/personal/x21234175_student_ncirl_ie/Documents/Recordings/Meeting%20with%20Donghyeok%20Lee-20230423_232922-Meeting%20Recording.mp4?web=1

1. Service Definitions

1.1. Domain Description

My domain is ‘Smart Warehouse (Track stock, automate order)’. In my project, smart warehouse means that it has 3 main services having rpc methods. The names of services are ‘InventoryManagementService’, ‘OrderManagementService’ and ‘TrackingDeliveryService’. In these services, there are 3, 3 and 2 RPC method each. Details for these will be explained ‘Service Implementations’ part.

Through these smart services, administrations can check the number of quantities for Item, update quantities, alert message for threshold, manage order status, delivery status and so on. Customer also can benefit by using these smart services: Create, cancel order and so on.

This domain is developed program language (Java) and uses open-source frame work ‘gRPC’ and other technical tools like JmDNS, Java swing and others.

1.2. Service Description

a) InventoryManagementService

This service provides 3 rpc methods (CheckItem, ModifyQuantity, and AlertOutOfStock). For database, CSV file is used. And With processing above 3 methods, the updated data record is also synchronized. This makes data be updated every time and avoid mismatching about inventory.

• **RPC Method 1 : CheckItem (Unary RPC)**

To check current quantities of specific itemID, user can use this method with itemID. And server will send message with the number of currentQuantities.

Request	Response
- itemID	- currentQuantities

• **RPC Method 2 : ModifyQuantity (Client Streaming RPC)**

To modify item’s quantities, user can use this method. If user request with itemID and updateQuantity. Server responds with success_failureMessage. Furthermore, because it’s implemented by Client Streaming RPC, user can choose how many item they want to modify with number

Request	Response
- itemID - updateQuantity	- success_failureMessage

- **RPC Method 3 : AlertOutOfStock (Server Streaming RPC)**

To check which item's quantity is less than specific number that is entered by user, user can use this method with threshold. Because it's implemented by server streaming RPC, if there are multiple dates less than threshold, server sends alertMessage many times.

Request	Response
- threshold	- alertMessage

b) OrderManagementService

This service provides 3 rpc methods (OrderItem, UpdateOrderStatus, and CancelOrder). For database, CSV file is used. And With processing above 3 methods, the updated data record is also synchronized. This makes data be updated every time and avoid mismatching about order information.

- **RPC Method 1 : OrderItem (Unary RPC)**

To place order item, the user can use this method with customerName, itemID and orderQuantities. In this method, there are some error handlings such as negative number of orderQuantities, server will respond if the request has no error.

Request	Response
- customerName - itemID - orderQuantities	- orderID - currentStatus - success_FailureMessage

- **RPC Method 2 : UpdateOrderStatus (Bi-Directional Streaming RPC)**

To modify orderStatus, user can use this method. Because it's implemented by Bi-Directional Streaming RPC, user can choose how many orderStatus they want to modify with number. And send request several times. Server is also send several messages after processing user's requests.

Request	Response
- orderID - newStatus	- orderID - currentStatus

- **RPC Method 3 : Cancel Order (Unary RPC)**

To cancel existing order, user can use this method. If user requests with orderID that they want to cancel, server deletes the order data matched with orderID requested by user. Lastly, server sends success_failureMessage.

Request	Response
- orderID	- success_failureMessage

- c) TrackingDeliveryService

This service provides 2 rpc methods (CheckShippingDetail, and UpdateShippingDetails). For database, CSV file is used. And With processing above 2 methods, the updated data record is also synchronized. This makes data be updated every time and avoid mismatching about order TrackingDelivery information.

- **RPC Method 1 : Check Shipping Details (Server Streaming RPC)**

To check existing shipping details, user can use this method. If user requests with customerName, server sends several messages matched with the customerName.

Request	Response
- customerName	- orderID - deliveryDetail - currentLocation - estimatedDeliveryDate

- **RPC Method 2 : Update Shipping Details (Client Streaming RPC)**

To update existing shipping details, user can use this method. If user requests with orderID and newDeliveryDetails, server sends success_failureMessage. Furthermore, this method is implemented by client streaming RPC, so user can choose the number of update existing shipping details at first.

Request	Response
- orderID - newDeliveryDetails	- success_failureMessage

2. Service Implementations

2.1. Overview

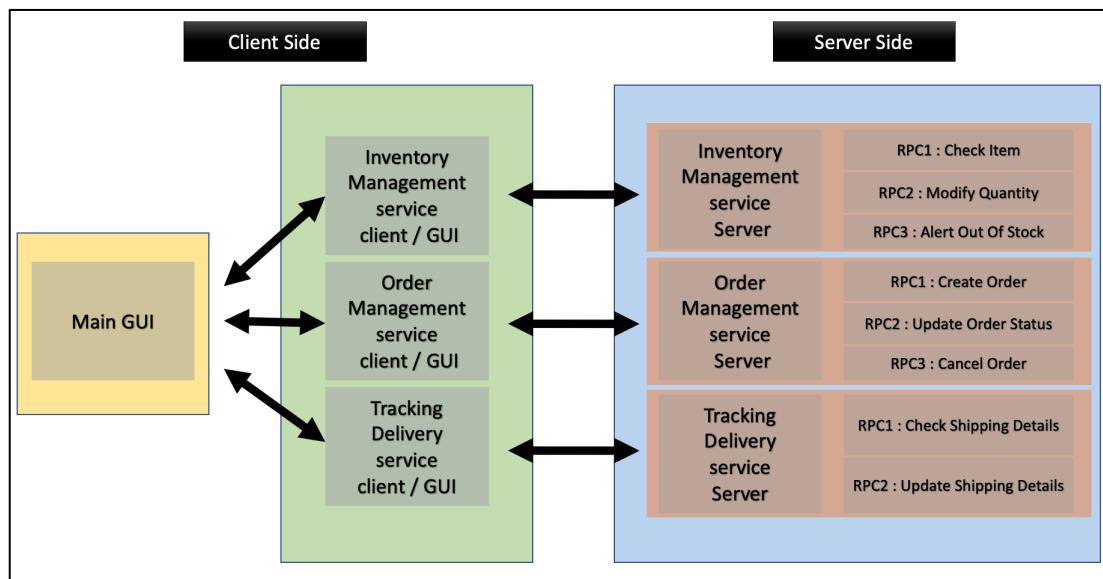
This service is implemented through gRPC. Put simply, there are 3 servers, 3 clients and 1 Main GUI (Integrate 3 services) and 3 clientGUIs. To implement this service, the required steps are like below.

- a) Run 3 servers with own host and port.
- b) Run ‘Main GUI’
- c) ‘Main GUI’ invokes each 3 clientGUIs according to user’s clicking service.
- d) Before clientGUI can be run, client side tries to retrieve ‘host’ and ‘port’ correct with its server by JmDNS.
- e) After retrieve host and port, clientGUI implements selected rpc method for user.

2.2. Domain and Service Structure

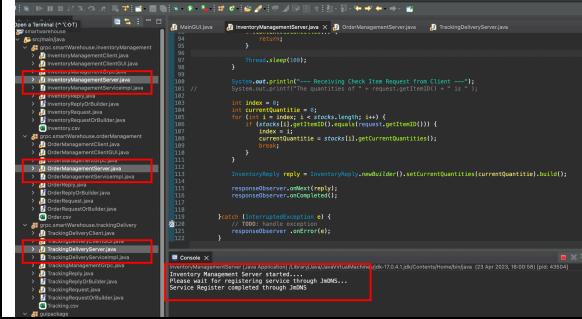
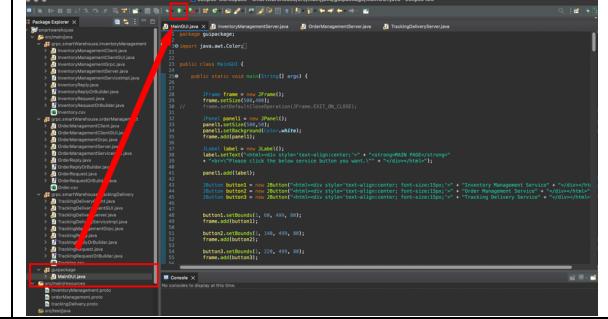
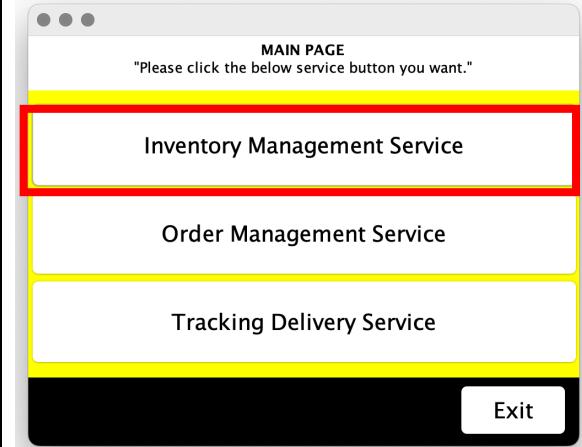
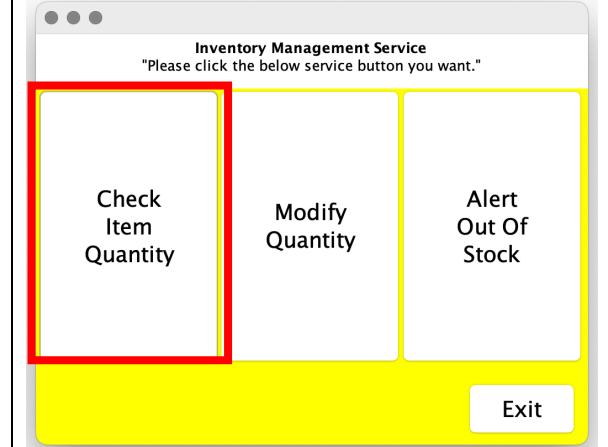
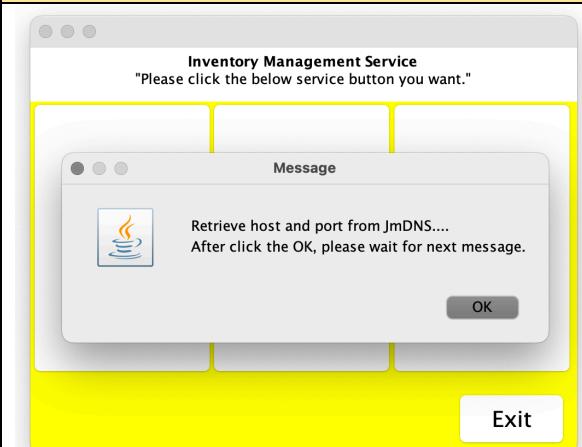
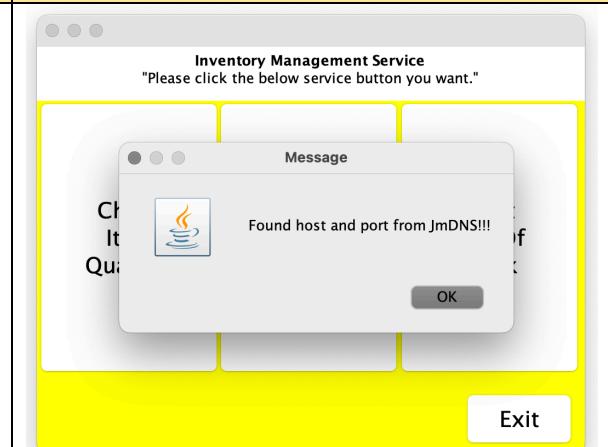
Brief domain structure is like below. User can use this domain by running Main GUI first. This main GUI is linked to 3 separate services’ GUI. And by clicking wanted RPC method, user can use RPC methods and interact with database in the server.

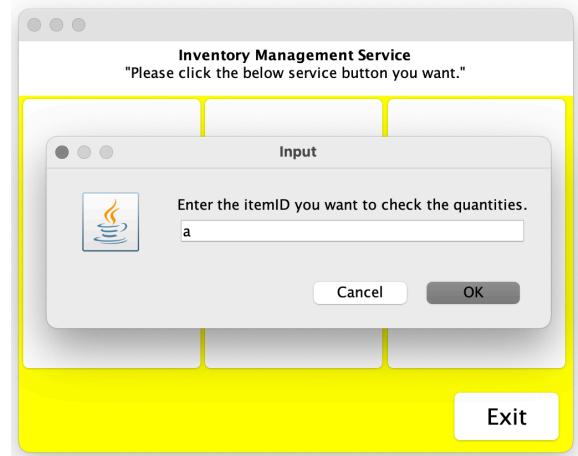
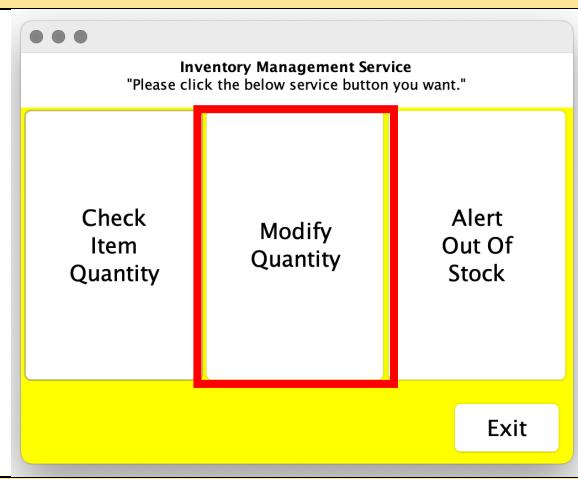
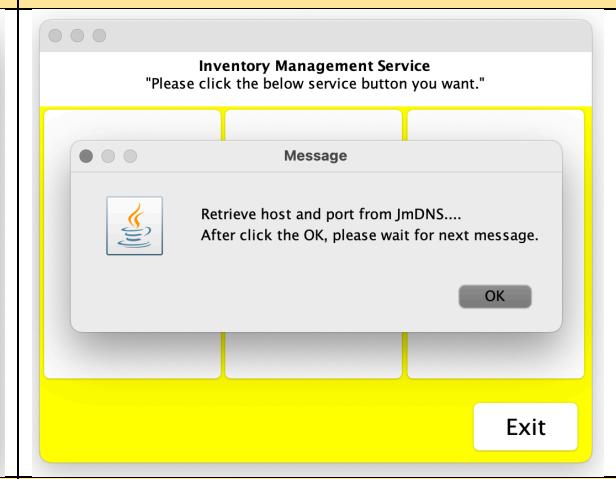
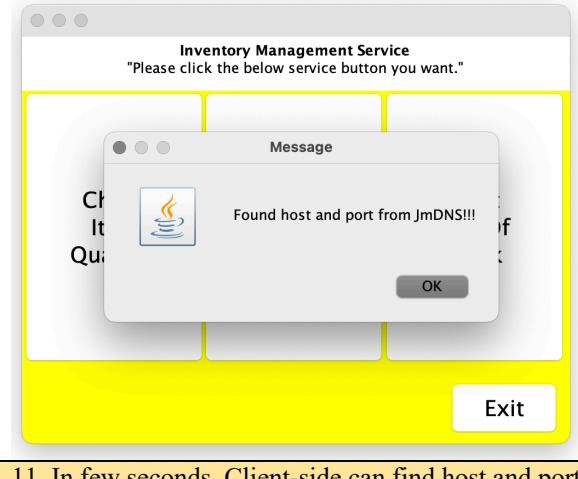
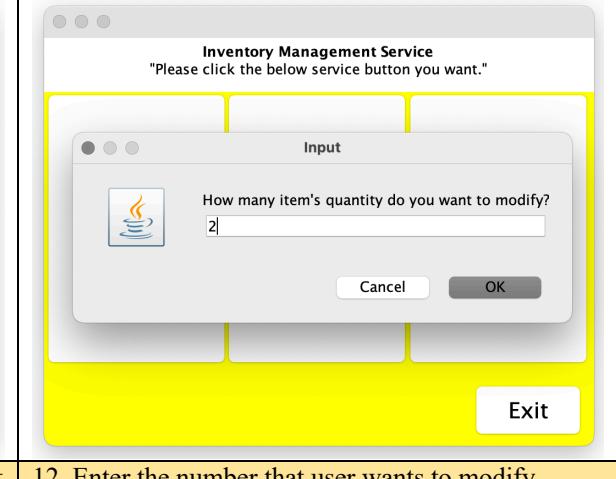
In this project, all database in the server side is implemented by CSV files. As user manipulate this data through RPC methods, database is updated spontaneously.

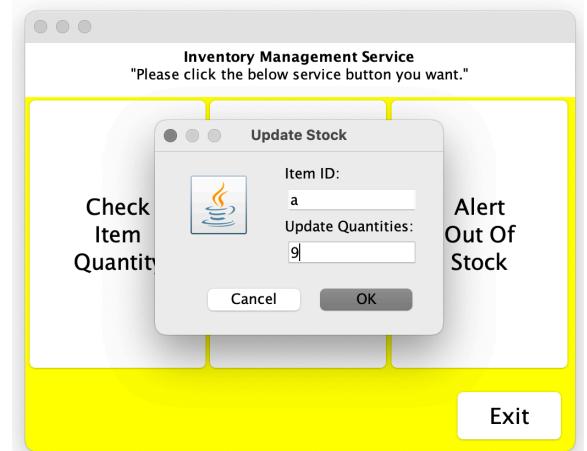
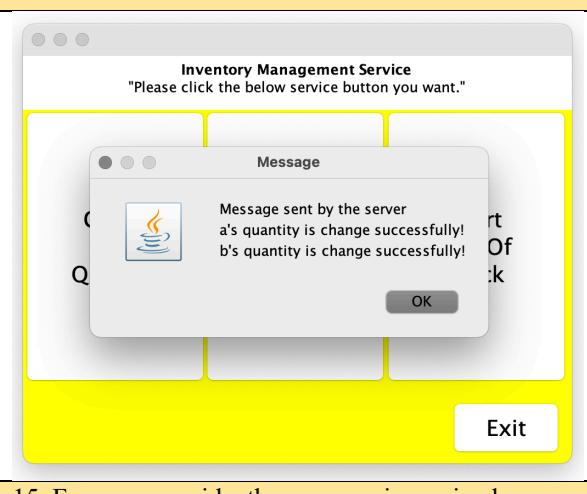
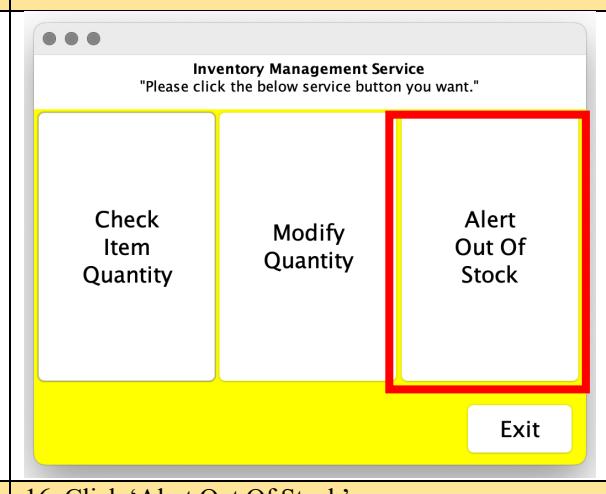
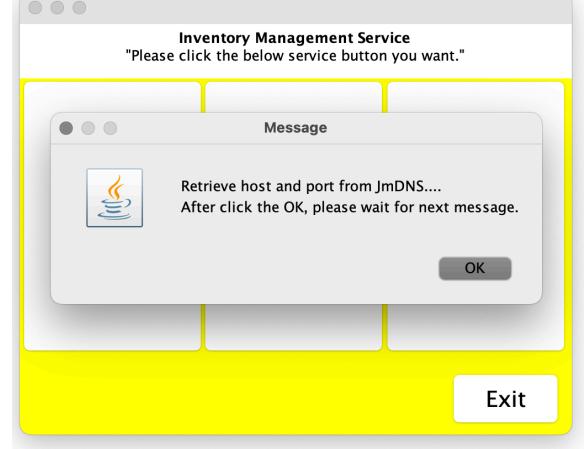


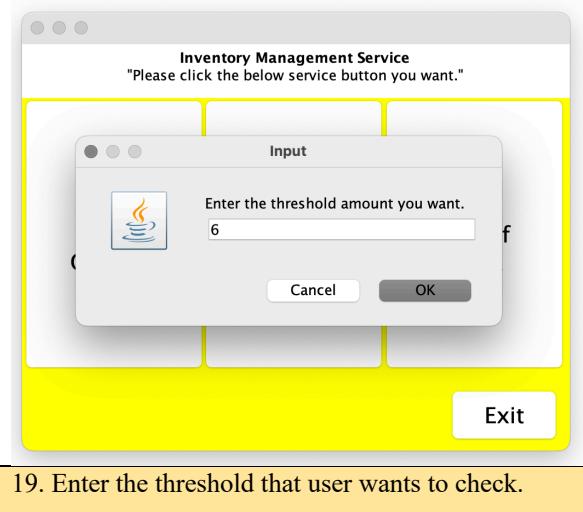
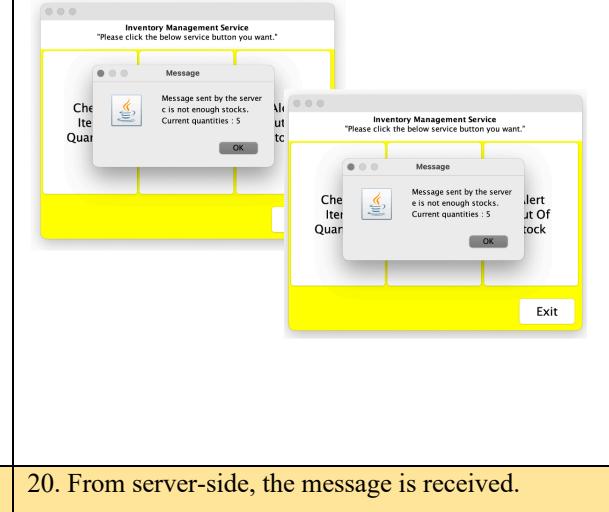
2.3. Service Implementation

a) InventoryManagementService

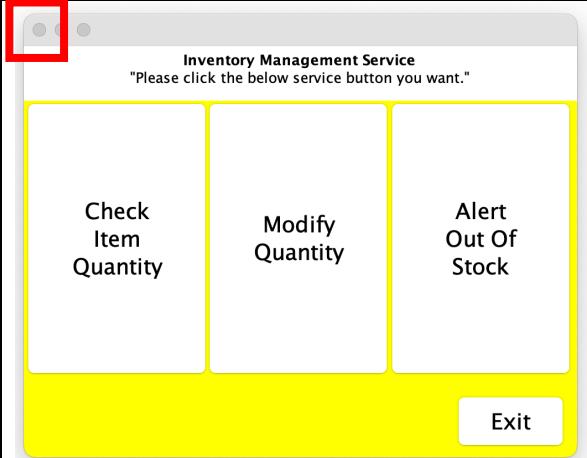
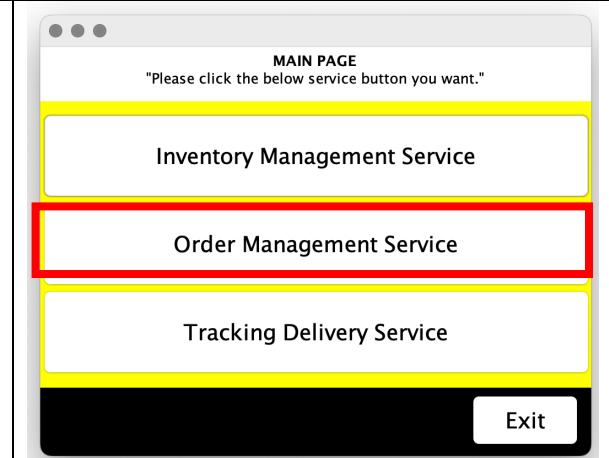
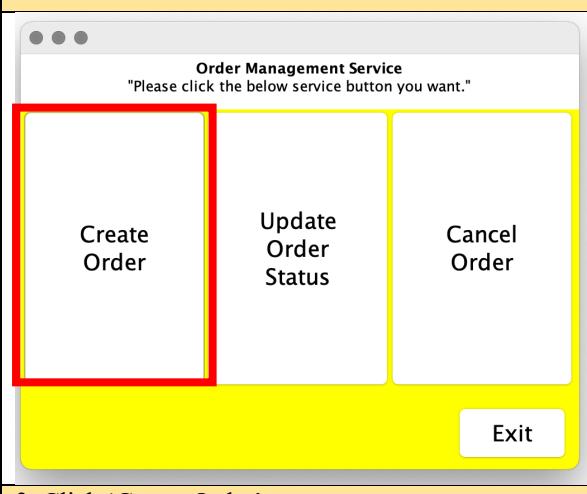
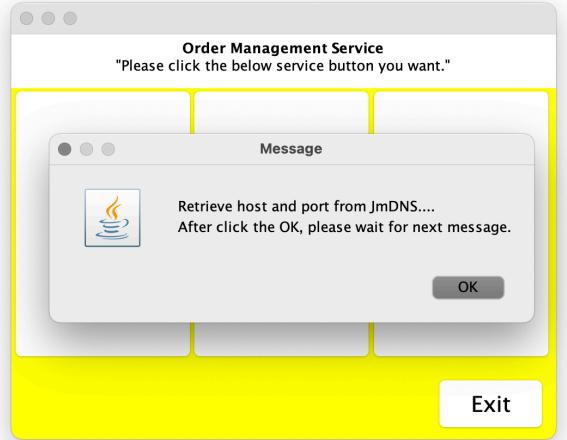
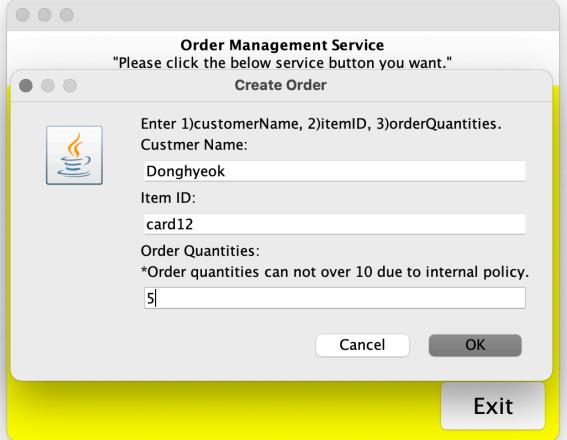
 <pre> ... System.out.println("--- Receiving Check Item Request from Client ---"); ... for (int i = index; i < stocks.length; i++) { if (stocks[i].currentQuantity > stocks[i].initialQuantity) { stocks[i].currentQuantity = stocks[i].getCurrentQuantities(); } } } InventoryReply reply = InventoryReply.newBuilder().setCurrentQuantities(currentQuantity).build(); responseObserver.onNext(reply); responseObserver.onCompleted(); } catch (InterruptedException e) { responseObserver.onError(e); } } </pre>	 <pre> import java.awt.Color; import java.awt.Container; import java.awt.GridLayout; import java.awt.event.ActionEvent; import java.awt.event.ActionListener; import javax.swing.JFrame; import javax.swing.JPanel; import javax.swing.JButton; import javax.swing.JLabel; import javax.swing.JOptionPane; import org.jmdns.JmDNS; import org.jmdns.JmDNSListener; import org.jmdns.services.SimpleService; import org.jmdns.services.SimpleServiceListener; public class Main { public static void main(String[] args) { JFrame frame = new JFrame(); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); JPanel panel = new JPanel(); panel.setLayout(new GridLayout(1, 0)); JButton button = new JButton("Inventory Management Service"); JButton button2 = new JButton("Order Management Service"); JButton button3 = new JButton("Tracking Delivery Service"); panel.add(button); panel.add(button2); panel.add(button3); frame.setContentPane(panel); frame.pack(); frame.setVisible(true); frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); JmDNS jmDNS = JmDNS.create("127.0.0.1"); jmDNS.addService("Inventory Management Service", "tcp", 12345, "com.example.InventoryManagementService"); jmDNS.addService("Order Management Service", "tcp", 12346, "com.example.OrderManagementService"); jmDNS.addService("Tracking Delivery Service", "tcp", 12347, "com.example.TrackingDeliveryService"); jmDNS.start(); } } </pre>
<p>1. First of all, execute servers. (Through JmDNS, all servers' services are registered)</p>	<p>2. Run the main GUI java file.</p>
	
<p>3. Click 'Inventory Management Service'.</p>	<p>4. Click 'Check Item Quantity'.</p>
	
<p>5. Client-side retrieve host and port from JmDNS</p>	<p>6. In few seconds, Client-side can find host and port from JmDNS.</p>

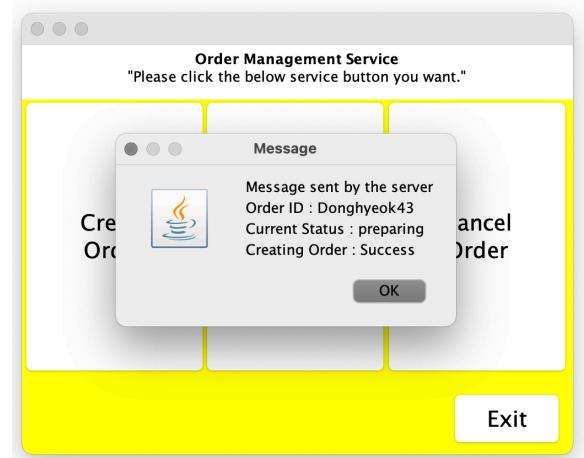
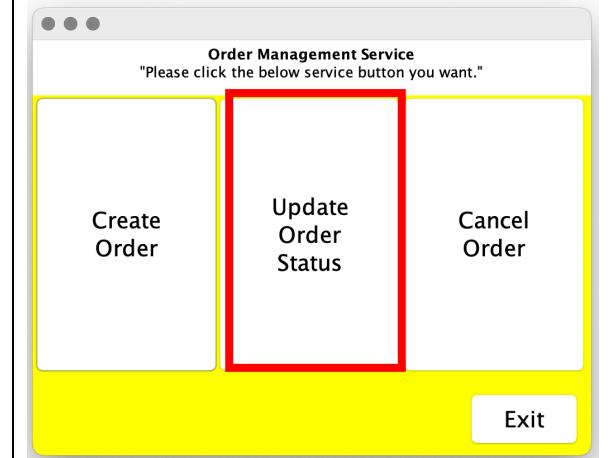
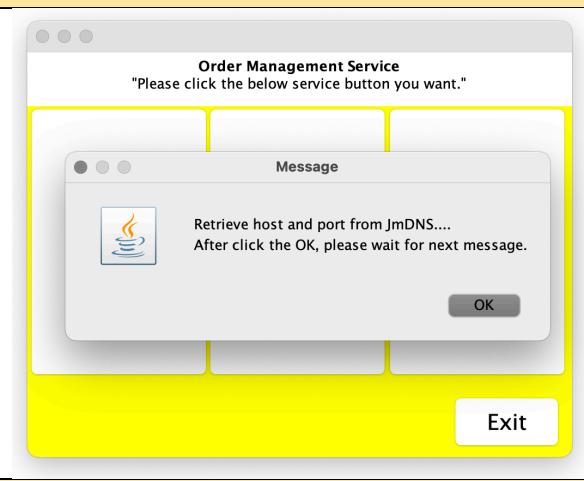
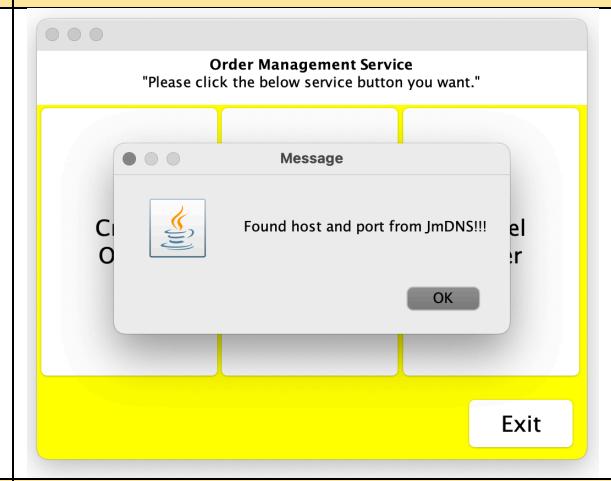
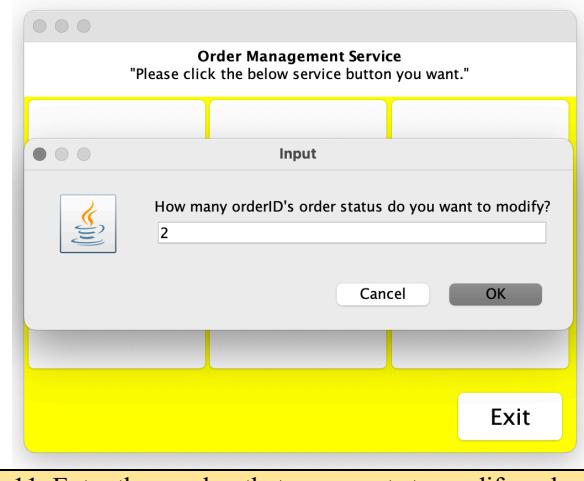
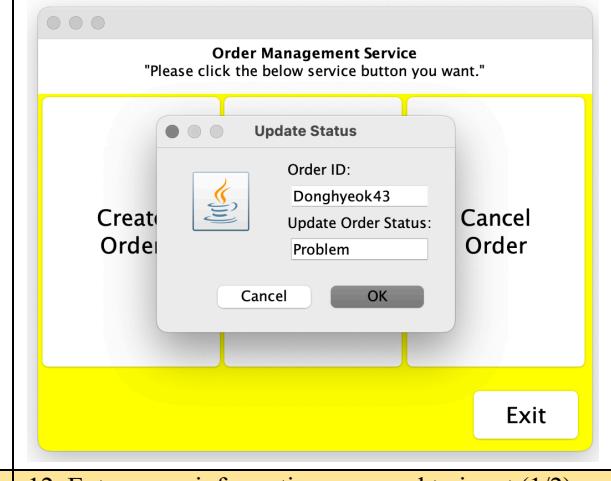
 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Input</p> <p>Enter the itemID you want to check the quantities. a</p> <p>Cancel OK</p>	 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Check Item Quantity The quantities of a : 5</p> <p>OK</p>
<p>7. Enter itemID user want to check the quantities.</p>	<p>8. From server-side, the message is received.</p>
 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Check Item Quantity Modify Quantity Alert Out Of Stock</p>	 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Retrieve host and port from JmDNS.... After click the OK, please wait for next message.</p> <p>OK</p>
<p>9. Click 'Modify Item Quantity'.</p>	<p>10. Client-side retrieve host and port from JmDNS.</p>
 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Found host and port from JmDNS!!!</p> <p>OK</p>	 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Input</p> <p>How many item's quantity do you want to modify? 2</p> <p>Cancel OK</p>
<p>11. In few seconds, Client-side can find host and port form JmDNS.</p>	<p>12. Enter the number that user wants to modify quantities.</p>

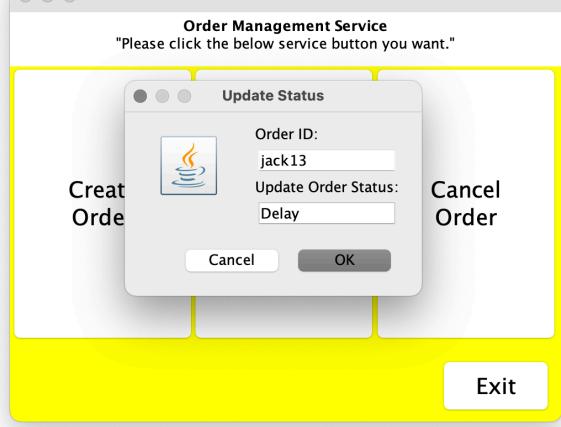
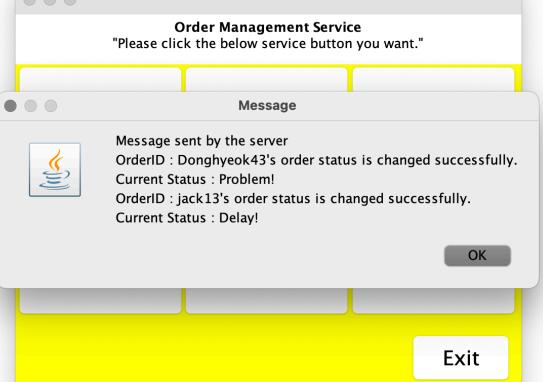
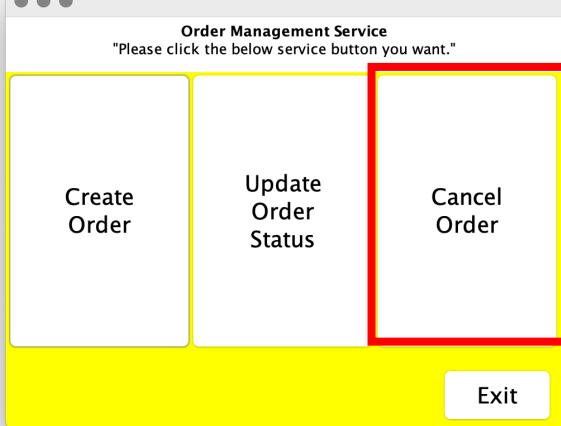
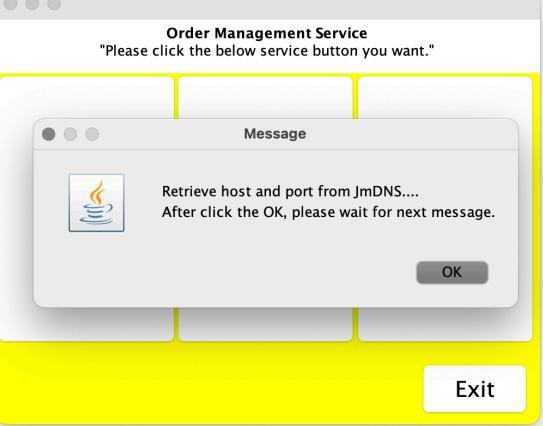
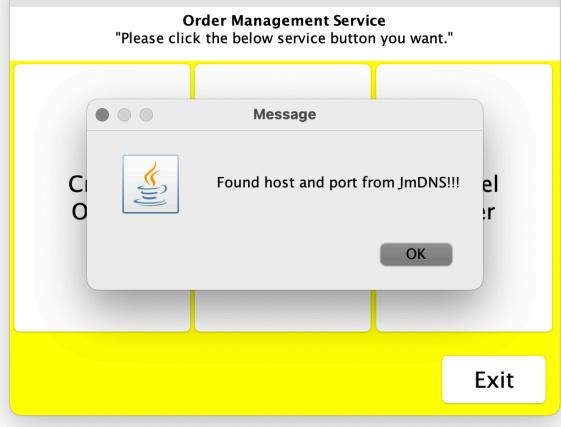
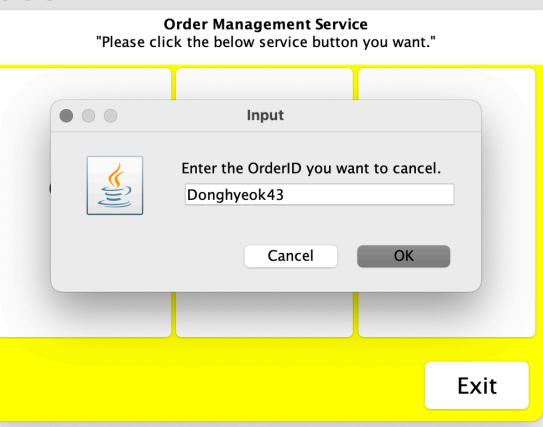
	
13. Enter the Item ID and Update Quantities (1/2)	14. Enter the Item ID and Update Quantities (2/2)
	
15. From server-side, the message is received.	16. Click 'Alert Out Of Stock'.
	
17. Client-side retrieve host and port from JmDNS.	18. In few seconds, Client-side can find host and port form JmDNS.

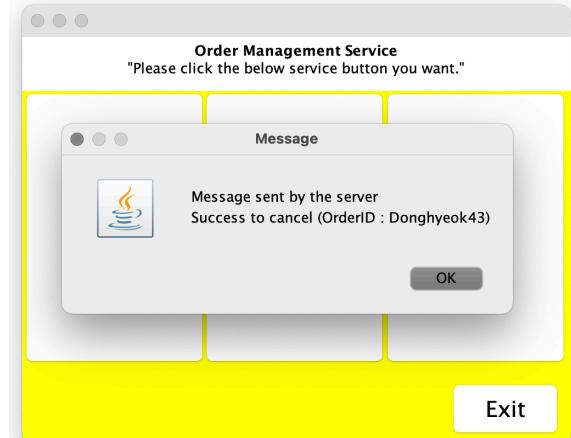
 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Input</p> <p>Enter the threshold amount you want.</p> <p>6</p> <p>Cancel OK</p> <p>Exit</p>	 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Check Item Quantity</p> <p>Message sent by the server : Stock level is not enough stocks. Current quantities : 5</p> <p>OK</p> <p>Inventory Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Check Item Quantity</p> <p>Message sent by the server : Stock level is not enough stocks. Current quantities : 5</p> <p>OK</p> <p>Alert Out Of Stock</p> <p>Exit</p>
<p>19. Enter the threshold that user wants to check.</p>	<p>20. From server-side, the message is received.</p>

b) OrderManagementService

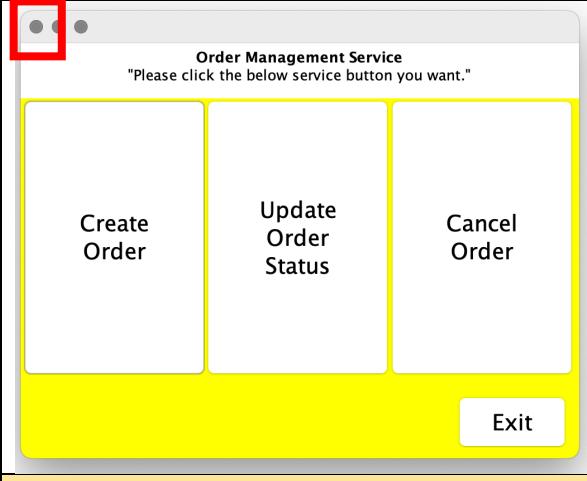
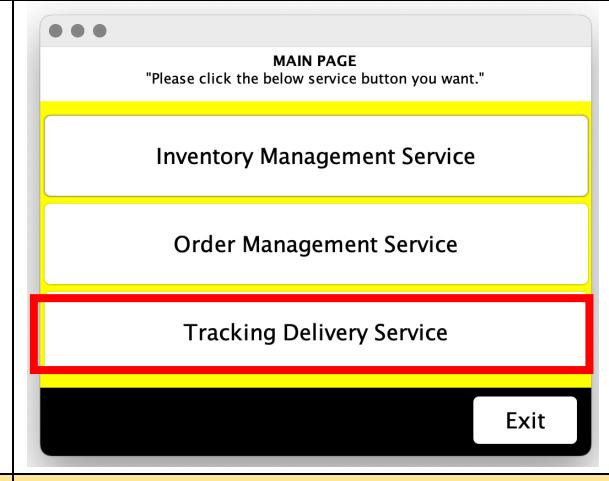
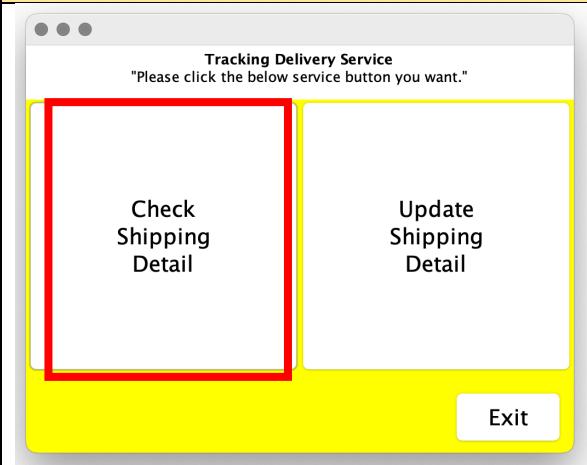
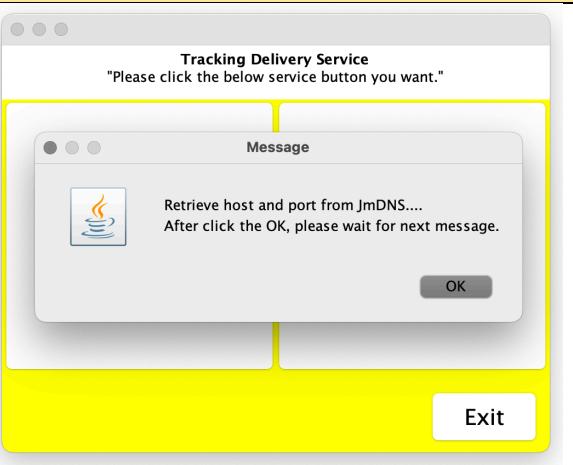
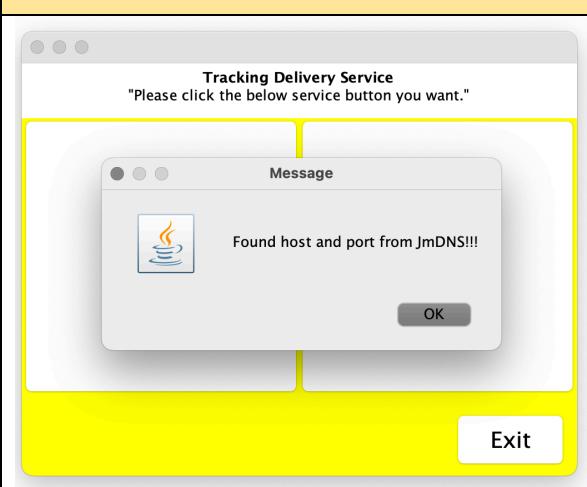
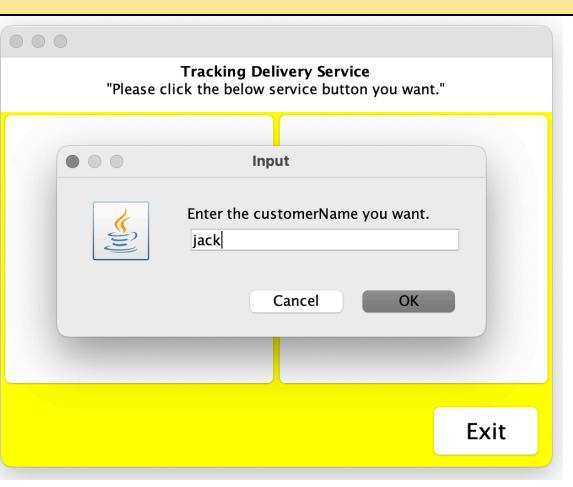
 <p>Inventory Management Service "Please click the below service button you want."</p> <p>Check Item Quantity Modify Quantity Alert Out Of Stock</p> <p>Exit</p>	 <p>MAIN PAGE "Please click the below service button you want."</p> <p>Inventory Management Service</p> <p>Order Management Service</p> <p>Tracking Delivery Service</p> <p>Exit</p>
<p>1. Click the 'X' button to return to first page.</p>  <p>Order Management Service "Please click the below service button you want."</p> <p>Create Order Update Order Status Cancel Order</p> <p>Exit</p>	<p>2. Click 'Order Management Service'.</p>  <p>Order Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Retrieve host and port from JmDNS.... After click the OK, please wait for next message.</p> <p>OK</p> <p>Exit</p>
<p>3. Click 'Create Order'.</p>  <p>Order Management Service "Please click the below service button you want."</p> <p>Create Order</p> <p>Message</p> <p>Found host and port from JmDNS!!!</p> <p>OK</p> <p>Exit</p>	<p>4. Client-side retrieve host and port from JmDNS</p>  <p>Order Management Service "Please click the below service button you want."</p> <p>Create Order</p> <p>Enter 1)customerName, 2)itemID, 3)orderQuantities.</p> <p>Custmer Name: Donghyeok</p> <p>Item ID: card12</p> <p>Order Quantities: *Order quantities can not over 10 due to internal policy. 5</p> <p>OK</p> <p>Cancel</p> <p>Exit</p>
<p>5. In few seconds, Client-side can find host and port from JmDNS.</p>	<p>6. Enter some information you need to input.</p>

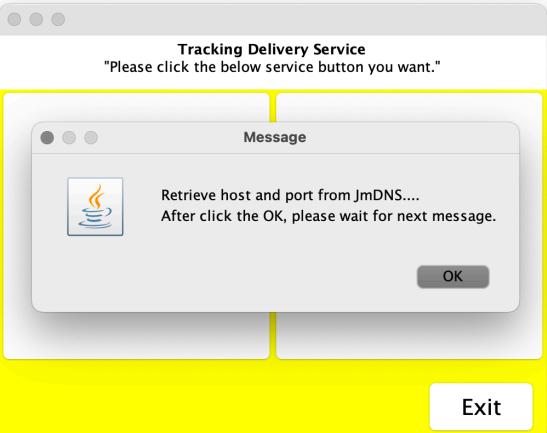
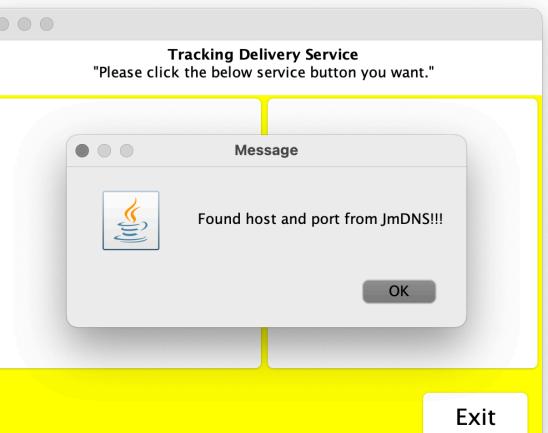
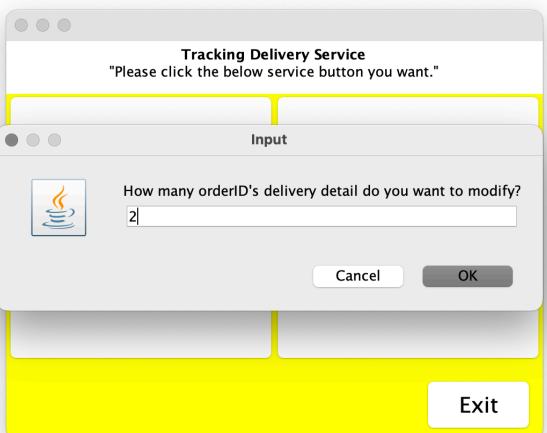
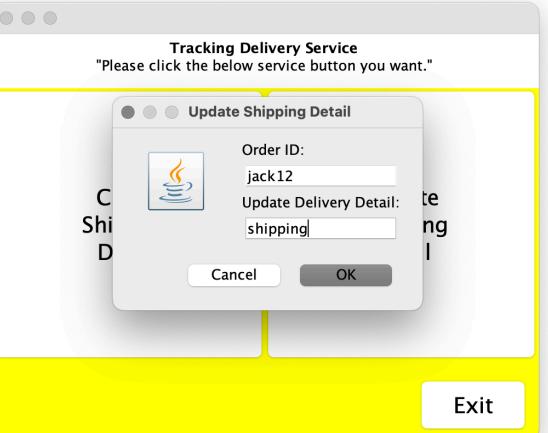
 <p>Order Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Message sent by the server Order ID : Donghyeok43 Current Status : preparing Creating Order : Success</p> <p>OK</p>	 <p>Order Management Service "Please click the below service button you want."</p> <p>Create Order Update Order Status Cancel Order</p> <p>Exit</p>
<p>7. From server-side, the message is received.</p>	<p>8. Click 'Modify Item Quantity'.</p>
 <p>Order Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Retrieve host and port from JmDNS.... After click the OK, please wait for next message.</p> <p>OK</p>	 <p>Order Management Service "Please click the below service button you want."</p> <p>Message</p> <p>Found host and port from JmDNS!!!</p> <p>OK</p>
<p>9. Client-side retrieve host and port from JmDNS.</p>	<p>10. In few seconds, Client-side can find host and port form JmDNS.</p>
 <p>Order Management Service "Please click the below service button you want."</p> <p>Input</p> <p>How many orderID's order status do you want to modify? 2</p> <p>Cancel OK</p>	 <p>Order Management Service "Please click the below service button you want."</p> <p>Update Status</p> <p>Order ID: Donghyeok43 Update Order Status: Problem</p> <p>Cancel OK</p>
<p>11. Enter the number that user wants to modify order status.</p>	<p>12. Enter some information you need to input (1/2).</p>

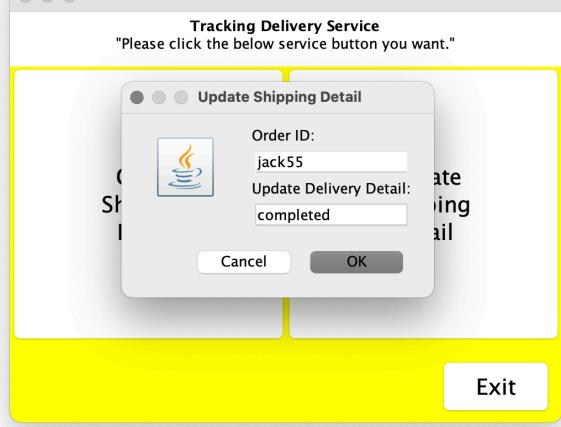
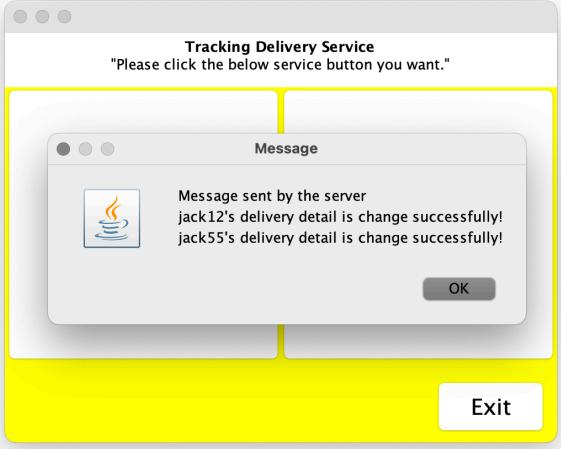
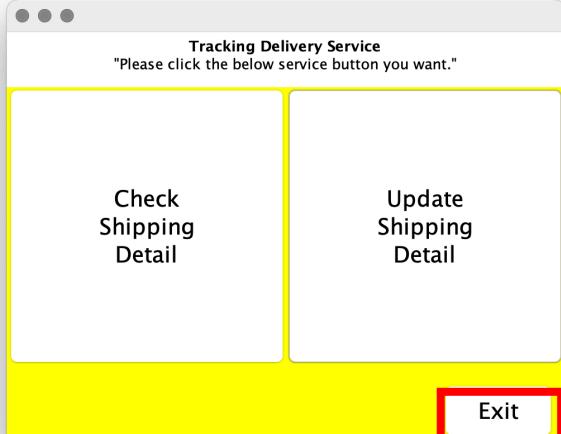
	
13. Enter some information you need to input (2/2).	14. From server-side, the message is received.
	
15. Click 'Modify Item Quantity'.	16. Client-side retrieve host and port from JmDNS.
	
17. In few seconds, Client-side can find host and port form JmDNS.	18. Enter OrderID user wants to cancel.

 <p>The screenshot shows a window titled "Order Management Service" with the sub-instruction "Please click the below service button you want.". A yellow rectangular box highlights a modal dialog titled "Message". Inside the dialog, there is a small icon of a coffee cup with steam, followed by the text "Message sent by the server" and "Success to cancel (OrderID : Donghyeok43)". At the bottom of the dialog are "OK" and "Exit" buttons. The entire highlighted area is yellow.</p>	
19. From server-side, the message is received.	

c) OrderManagementService

	
<p>1. Click the 'X' button to return to first page.</p> 	<p>2. Click 'Tracking Delivery Service'.</p> 
<p>3. Click 'Check Shipping Detail'.</p> 	<p>4. Client-side retrieve host and port from JmDNS.</p> 
<p>5. In few seconds, Client-side can find host and port form JmDNS.</p>	<p>6. Enter the customerName that user wants to check.</p>

	
<p>7. From server-side, the message is received.</p>	<p>8. Click 'Update Shipping Detail'.</p>
	
<p>9. Client-side retrieve host and port from JmDNS.</p>	<p>10. In few seconds, Client-side can find host and port form JmDNS.</p>
	
<p>11. Enter the number that user wants to modify delivery detail.</p>	<p>12. Enter some information you need to input (1/2).</p>

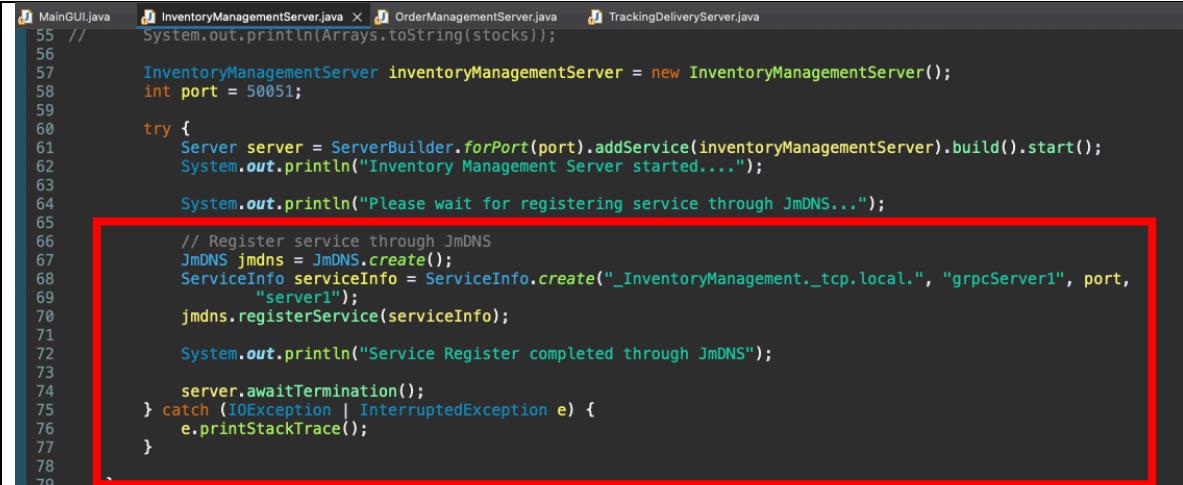
	
13. Enter some information you need to input (2/2).	14. From server-side, the message is received.
	
15. Click the 'Exit' button to terminate program.	

3. Naming Service

3.1. Overview

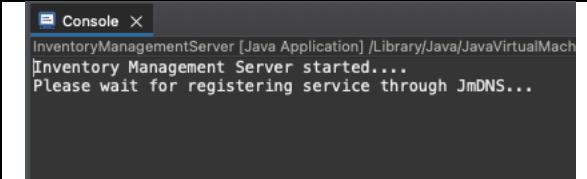
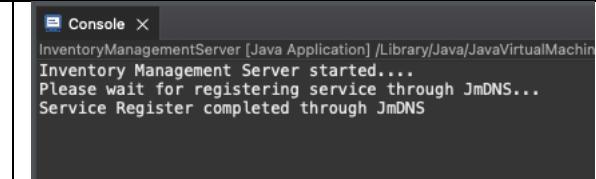
Naming services in gRPC is meaning that retrieving proper server through services' name. Naming service transfers service's name into IP address and port. In the distribute systems, this service is core skill to link correct service to client-side. In this project, I also use a Naming service : Java Multicast DNS (jmdns) and implement all client-sides through jmdns to discover correct service in server-side. The code related to jmdns is like below.

a) InventoryManagementService



```
1 MainGUI.java 2 InventoryManagementServer.java X 3 OrderManagementServer.java 4 TrackingDeliveryServer.java
55 // 56 System.out.println(Arrays.toString(stocks));
57 InventoryManagementServer inventoryManagementServer = new InventoryManagementServer();
58 int port = 50051;
59
60 try {
61     Server server = ServerBuilder.forPort(port).addService(inventoryManagementServer).build().start();
62     System.out.println("Inventory Management Server started....");
63
64     System.out.println("Please wait for registering service through JmDNS...");
65
66     // Register service through JmDNS
67     JmDNS jmDNS = JmDNS.create();
68     ServiceInfo serviceInfo = ServiceInfo.create("_InventoryManagement._tcp.local.", "grpcServer1", port,
69             "server1");
70     jmDNS.registerService(serviceInfo);
71
72     System.out.println("Service Register completed through JmDNS");
73
74     server.awaitTermination();
75 } catch (IOException | InterruptedException e) {
76     e.printStackTrace();
77 }
```

Code to register service through JmDNS in Inventory Management Server

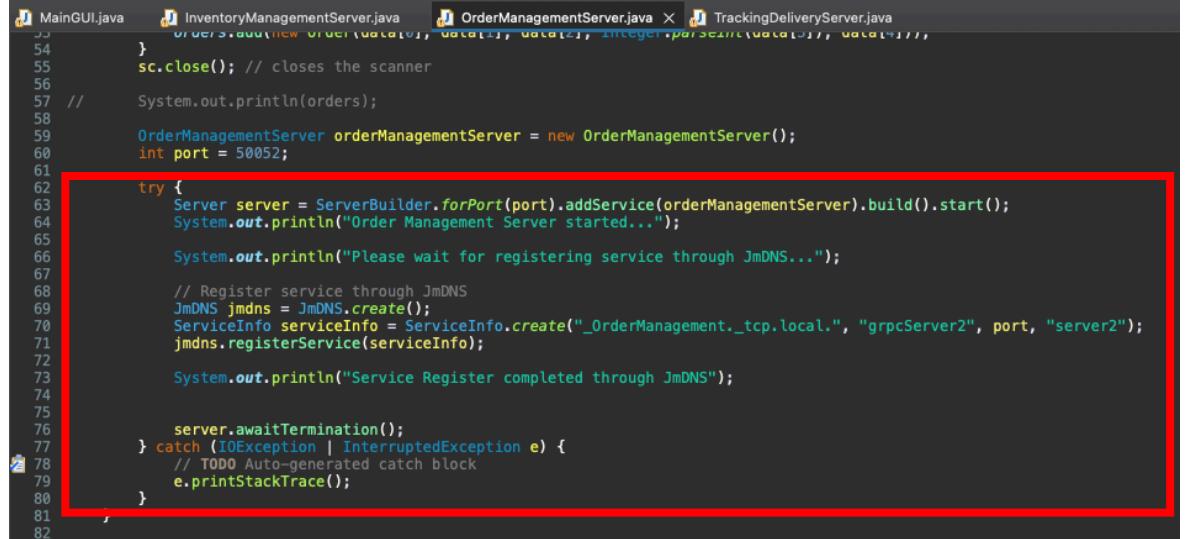
	
1. Process to register...	2. Finish to register.

```
1 public static void main(String[] args) throws InterruptedException, IOException {
2
3     // System.out.println("Retrieve host and port from JmDNS....");
4     JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the OK, please wait for next message.");
5
6     // Receive server information through JmDNS
7     JmDNS jmDNS = JmDNS.create();
8     ServiceInfo[] services = jmDNS.list("_InventoryManagement._tcp.local.");
9     if (services.length == 0) {
10         System.out.println("No gRPC server found");
11         JOptionPane.showMessageDialog(null, "No gRPC server found");
12         return;
13     }
14
15     // Receive host and port through gRPC
16     ServiceInfo serviceInfo = services[0];
17     host = serviceInfo.getHostAddresses()[0];
18     port = serviceInfo.getPort();
19
20     // System.out.println("Found host and port from JmDNS!!!!");
21     JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!!");
22 }
```

Client found service through JmDNS

b) OrderManagementService

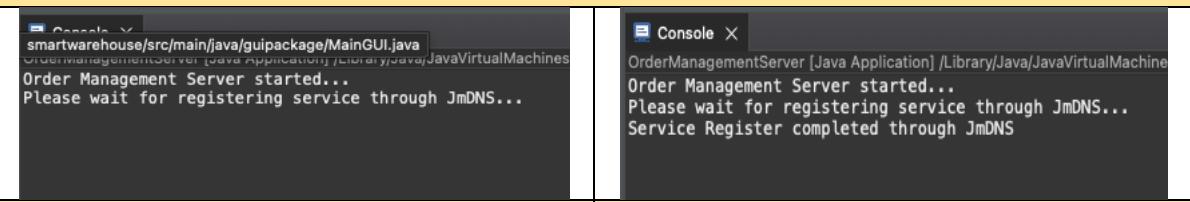
Code to register service through JmDNS in Order Management Server



```

54     }
55     sc.close(); // closes the scanner
56
57 //    System.out.println(orders);
58
59     OrderManagementServer orderManagementServer = new OrderManagementServer();
60     int port = 50052;
61
62     try {
63         Server server = ServerBuilder.forPort(port).addService(orderManagementServer).build().start();
64         System.out.println("Order Management Server started...");
65
66         System.out.println("Please wait for registering service through JmDNS...");
67
68         // Register service through JmDNS
69         JmDNS jmDNS = JmDNS.create();
70         ServiceInfo serviceInfo = ServiceInfo.create("_OrderManagement._tcp.local.", "grpcServer2", port, "server2");
71         jmDNS.registerService(serviceInfo);
72
73         System.out.println("Service Register completed through JmDNS");
74
75         server.awaitTermination();
76     } catch (IOException | InterruptedException e) {
77         // TODO Auto-generated catch block
78         e.printStackTrace();
79     }
80 }
```

1. Process to register... 2. Finish to register.



```

smartwarehouse@smartwarehouse:~/src/main/java/guipackage/MainGUI.java
OrderManagementServer [Java Application] /Library/Java/JavaVirtualMachines
Order Management Server started...
Please wait for registering service through JmDNS...

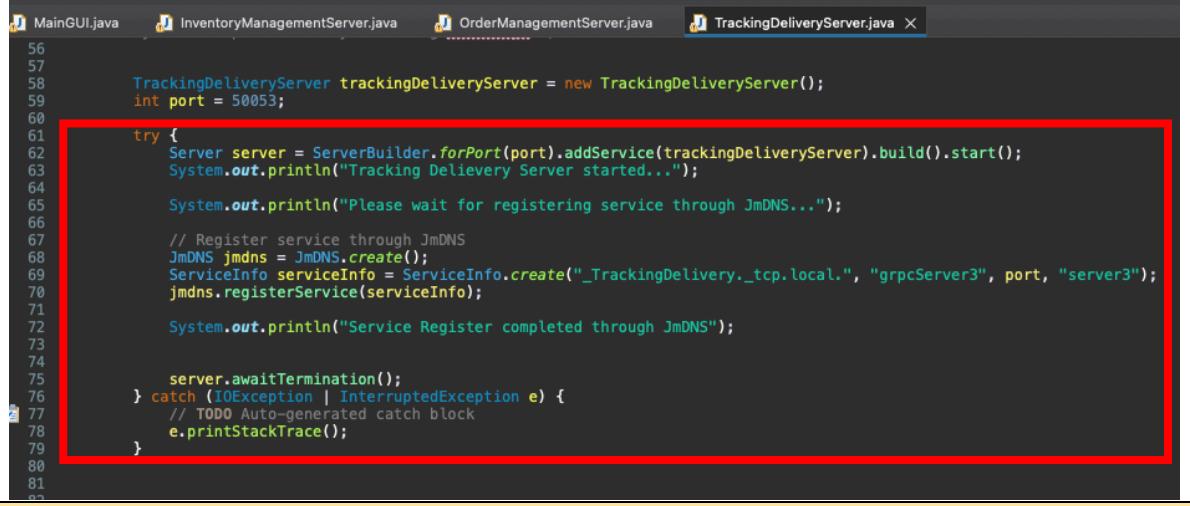
smartwarehouse@smartwarehouse:~/src/main/java/guipackage/MainGUI.java
OrderManagementServer [Java Application] /Library/Java/JavaVirtualMachine
Order Management Server started...
Please wait for registering service through JmDNS...
Service Register completed through JmDNS
```

Client found service through JmDNS

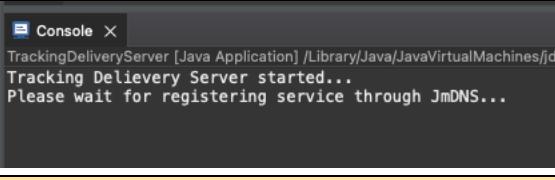
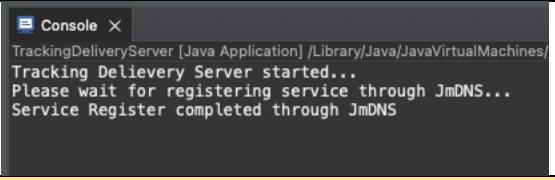
```

36 //    System.out.println("Retrieve host and port from JmDNS....");
37 JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the OK, please wait for next message.");
38
39
40 // Receive server information through JmDNS
41 JmDNS jmDNS = JmDNS.create();
42 ServiceInfo[] services = jmDNS.list("_OrderManagement._tcp.local.");
43 if (services.length == 0) {
44     System.out.println("No gRPC server found");
45     JOptionPane.showMessageDialog(null, "No gRPC server found");
46     return;
47 }
48
49 // Receive host and port through gRPC
50 ServiceInfo serviceInfo = services[0];
51 host = serviceInfo.getHostAddresses()[0];
52 port = serviceInfo.getPort();
53
54 //    System.out.println("Found host and port from JmDNS!!!!");
55 JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!!");
56 }
```

c) TrackingDeliveryService



Code to register service through JmDNS in Tracking Delivery Server

	
---	--

1. Process to register... 2. Finish to register.



Client found service through JmDNS

4. Error Handling & Advanced Features

4.1. Error Handling

In the project, I use some try-catch statement to handle errors and if / else-if / while loop to guide client to enter correct input that can be received by server-side. One of error handling code is like below.

- **Input negative value for ‘orderQuantities’ in ‘Create order’ method of Order Management Service.**

The screenshot shows three tabs in a Java IDE: MainGUI.java, InventoryManagementServer.java, and OrderManagementServer.java. The OrderManagementServer.java tab is active, displaying code for an RPC method. A red box highlights the error handling logic. The code checks if the order quantity is less than zero, sends an error response with a description, and returns. Below the IDE is a yellow bar labeled "Error Handling in Order Management Server".

```
82  
83 // RPC Method 1 : Create Order (Unary RPC)  
84  
85● @Override  
86     public void orderItem(OrderRequest request, StreamObserver<OrderReply> responseObserver) {  
87         // TODO Auto-generated method stub  
88  
89         System.out.println("---- Receiving Order Item Request from Client ----");  
90  
91         // Error handling Implementation  
92         if(request.getOrderQuantities() < 0) {  
93             responseObserver.onError(Status.INVALID_ARGUMENT  
94                     .withDescription("The order quantity being sent cannot be negative")  
95                     .augmentDescription("Order Quantity : " + request.getOrderQuantities())  
96                     .asRuntimeException());  
97         }  
98         return;  
99     }  
100  
101
```

The screenshot then transitions to the OrderManagementClient.java tab, also with a red box highlighting the error handling logic. It shows code for creating an order, sending a request, and catching a runtime exception. Below the IDE is a yellow bar labeled "Error Handling in Order Management Client".

```
77  
78     // RPC Method 1 : Create Order (Unary RPC)  
79●     private static void createOrder(ManagedChannel channel) {  
80         System.out.println("Enter createOrder");  
81         OrderManagementBlockingStub blockingStub = OrderManagementGrpc.newBlockingStub(channel);  
82  
83         System.out.println("Enter 1)customerName, 2)itemID, 3)orderQuantities. *Order quantities can not over 10 due to internal policy.");  
84         String customerName = sc.next();  
85         String itemID = sc.next();  
86         String orderQuantities = sc.next();  
87  
88         //Error implementing check  
89         try {  
90             OrderRequest request = OrderRequest.newBuilder()  
91                     .setCustomerName(customerName)  
92                     .setItemID(itemID)  
93                     .setOrderQuantities(orderQuantities)  
94                     .build();  
95  
96             OrderReply reply = blockingStub.orderItem(request);  
97  
98             System.out.println("Message sent by the server ");  
99             System.out.println(reply.getOrderId());  
100            System.out.println(reply.getCurrentStatus());  
101            System.out.println(reply.getSuccessFailureMessage());  
102  
103        } catch (RuntimeException e) {  
104            // TODO: handle exception  
105            System.out.println("Got an Exception for order quantity" + "\nOrder Quantity : " + orderQuantities);  
106            e.printStackTrace();  
107        }  
108    }  
109  
110  
111
```

Finally, a terminal window titled "Console" shows the execution of the client code. It prints host and port information, prompts for order details, and then displays an error message about order quantities. Below the terminal is a yellow bar labeled "Enter '-1' for orderQuantities (Improper input)".

```
Console  
OrderManagementClient_createOrder [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java (23 Apr 2023, 20:26)  
Retrieve host and port from JmDNS....  
Found host and port from JmDNS!!!  
Enter createOrder  
Enter 1)customerName, 2)itemID, 3)orderQuantities. *Order quantities can not over 10 due to internal policy.  
donghyeok safai -1|
```

Enter '-1' for orderQuantities (Improper input)

```

Console X
OrderManagementClient_createOrder [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java (23 Apr 2023, 20:26:24) [pid: 45190]
Enter 1)customerName, 2)itemID, 3)orderQuantities. *Order quantities can not over 10 due to internal policy.
donghyeok safai -1
Got an Exception for order quantity
Order Quantity : -1
io.grpc.StatusRuntimeException: INVALID_ARGUMENT: The order quantity being sent cannot be negative
Order Quantity : -1
    at io.grpc.stub.ClientCalls.toStatusRuntimeException(ClientCalls.java:233)
    at io.grpc.stub.ClientCalls.getUnchecked(ClientCalls.java:214)
    at io.grpc.stub.ClientCalls.blockingUnaryCall(ClientCalls.java:139)
    at grpc.smartWarehouse.orderManagement.OrderManagementGrpc$OrderManagementBlockingStub.orderItem(OrderManagementGrpc.java:286)
    at grpc.smartWarehouse.orderManagement.OrderManagementClient.createOrder(OrderManagementClient.java:97)
    at grpc.smartWarehouse.orderManagement.OrderManagementClient.main(OrderManagementClient.java:65)

```

Server-side sends detail of error (INVAID_ARGUMENT)

4.2. Deadlines

In the project, I also use some of advanced features like deadline. Through deadline, client can give limited time that response will be processed in this time. One of the codes that uses deadline feature is like below.

- **Deadline in Inventory Management Server and Client-side**

```

MainGUI.java  InventoryManagementServer.java  OrderManagementServer.java  TrackingDeliveryServer.java  InventoryManagementClient.java
79 }
80
81 // RPC Method 1 : Check Item (Unary RPC)
82
83● @Override
84 public void checkItem(InventoryRequest request, StreamObserver<InventoryReply> responseObserver) {
85     // TODO Auto-generated method stub
86
87     // Deadline
88     Context context = Context.current();
89
90     try {
91         for (int i = 0; i < 3; ++i) {
92             if (context.isCancelled()) {
93                 return;
94             }
95
96             Thread.sleep(100);
97         }
98
99         System.out.println("--- Receiving Check Item Request from Client ---");
100
101        int index = 0;
102        int currentQuantitie = 0;
103        for (int i = index; i < stocks.length; i++) {
104            if (stocks[i].getItemID().equals(request.getItemID())) {
105                index = i;
106                currentQuantitie = stocks[i].getCurrentQuantities();
107            }
108        }
109    }
110
111    responseObserver.onNext(InventoryReply.newBuilder().setStockId(stocks[index].getStockId())
112                            .setStockName(stocks[index].getStockName())
113                            .setStockCurrentQuantities(currentQuantitie)
114                            .build());
115    responseObserver.onCompleted();
116}

```

After executing checkItem RPC, server implement Thread.sleep(100) 3 times.

```

MainGUI.java  InventoryManagementServer.java  OrderManagementServer.java  TrackingDeliveryServer.java  InventoryManagementClient.java
87
88 // RPC Method 1 : Check Item (Unary RPC)
89● private static void checkItem(ManagedChannel channel) {
90     System.out.println("Enter checkItem");
91     InventoryManagementBlockingStub blockingStub = InventoryManagementGrpc.newBlockingStub(channel);
92     InventoryManagementGrpc.InventoryManagementBlockingStub stub = InventoryManagementGrpc.newBlockingStub(channel);
93
94     System.out.println("Enter the itemID you want to check the quantities.");
95     String setItemID = sc.nextLine();
96
97     InventoryRequest request = InventoryRequest.newBuilder().setItemID(setItemID).build();
98
99    // InventoryReply reply = blockingStub.checkItem(request);
100
101    // Deadline not exceed case
102    InventoryReply reply = stub.withDeadline(deadline.after(3, TimeUnit.SECONDS)).checkItem(request);
103
104    System.out.println("Message sent by the server ");
105
106    System.out.print("The quanties of " + request.getItemID() + " : ");
107    System.out.println(reply.getCurrentQuantities());
108
109}

```

In client-side, deadline.after is 3 seconds, so it's much than Thread sleep time in server.

```
Console X
InventoryManagementClient_checkitem [Java Application] [pid: 45339]
Retrieve host and port from JmDNS....
Found host and port from JmDNS!!!
Enter checkItem
Enter the itemID you want to check the quantities.
a
Message sent by the server
The quanties of a : 5
```

So, server sends message normally.

```
109
110
111
112    // Deadline exceeded case
113
114    try {
115        InventoryReply reply = stub.withDeadline(deadline.after(100, TimeUnit.MILLISECONDS)).checkItem(request);
116        System.out.println("Within DeadLine" + reply.getCurrentQuantities());
117    } catch (StatusRuntimeException e) {
118        // TODO: handle exception
119        if(e.getStatus().getCode() == Status.Code.DEADLINE_EXCEEDED) {
120            System.out.println("Deadline has been exceeded");
121        } else {
122            System.out.println("Got an exception in checkItem");
123            e.printStackTrace();
124        }
125    }
126
127
128
```

In case of deadline.after 100 MILLISECONDS, it's less than Thread sleep time in server

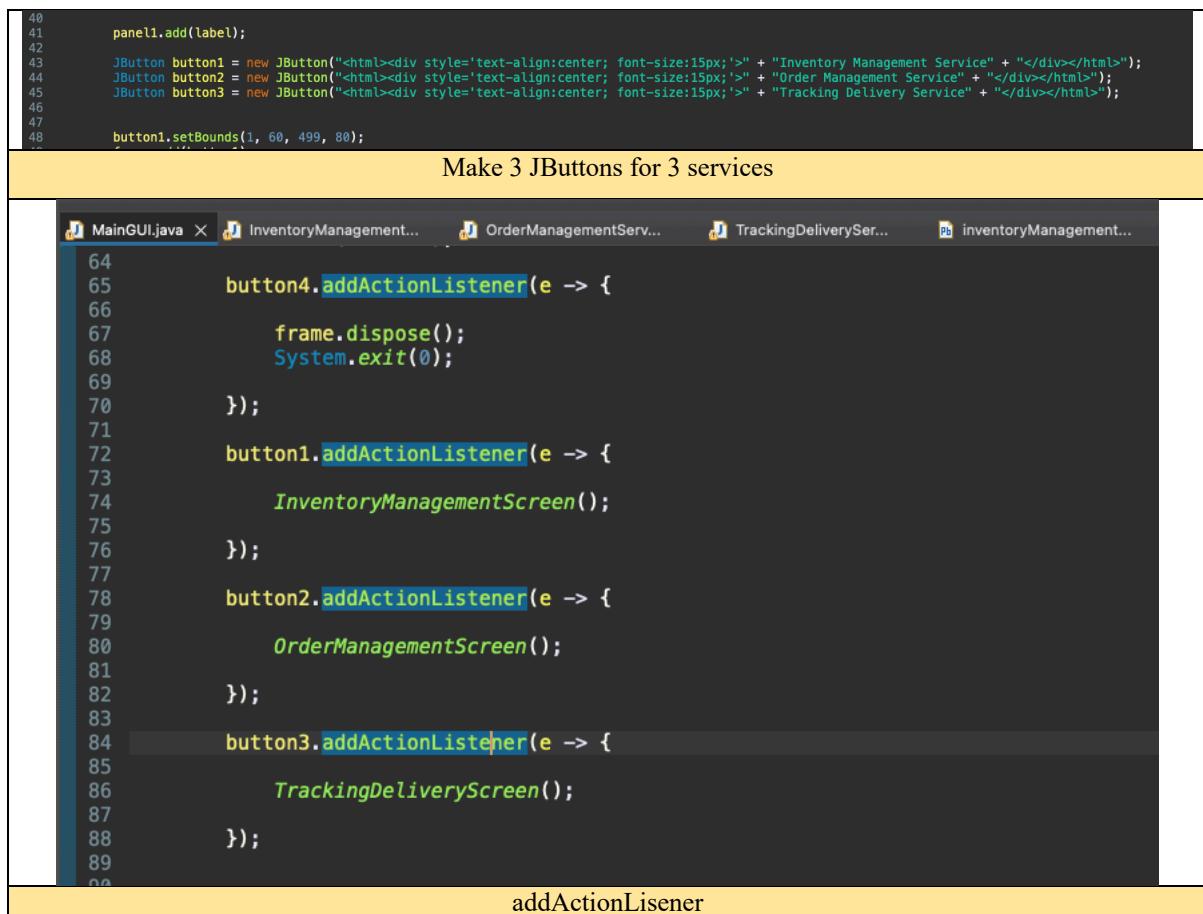
```
Console X
InventoryManagementClient_checkitem [Java Application] /Library/Java/JavaVirtualM
Retrieve host and port from JmDNS....
Found host and port from JmDNS!!!
Enter checkItem
Enter the itemID you want to check the quantities.
a
Deadline has been exceeded
```

So, server can't send message like normal.

5. Client – Graphical User Interface (GUI)

5.1. mainGUI

GUI Interface was explained in service implementation part. In this part, I explain how to make the GUI and which concept was used for. I used Java Swing to implement GUI. And because there are 3 services, I made 3 JButtons and this buttons have function through ‘addActionListener’ in mainGUI.



```
40     panel1.add(label);
41
42
43     JButton button1 = new JButton("<html><div style='text-align:center; font-size:15px;'>" + "Inventory Management Service" + "</div></html>");
44     JButton button2 = new JButton("<html><div style='text-align:center; font-size:15px;'>" + "Order Management Service" + "</div></html>");
45     JButton button3 = new JButton("<html><div style='text-align:center; font-size:15px;'>" + "Tracking Delivery Service" + "</div></html>");
46
47
48     button1.setBounds(1, 60, 499, 80);
```

Make 3 JButtons for 3 services

```
64         button4.addActionListener(e -> {
65
66             frame.dispose();
67             System.exit(0);
68         });
69
70         button1.addActionListener(e -> {
71
72             InventoryManagementScreen();
73         });
74
75         button2.addActionListener(e -> {
76
77             OrderManagementScreen();
78         });
79
80         button3.addActionListener(e -> {
81
82             TrackingDeliveryScreen();
83         });
84
85     });
86
87     button3.addActionListener(e -> {
88
89         TrackingDeliveryScreen();
90     });
91
92 }
```

addActionLisener

5.2. 3 GUIs in client-side

After making main GUI, for linking to each services’ RPCs, I made 3 single GUI for client-side. And main GUI implementing way in the single GUI is JOptionPane of Java Swing.



```
69         switch (args[0]) {
70             case "checkItem":
71                 checkItem(channel);
72                 break;
73             case "modifyQuantity":
74                 modifyQuantity(channel);
75                 break;
76             case "alertOutOfStock":
77                 alertOutOfStock(channel);
78                 break;
79             default:
80                 System.out.println("Keyword Invalid " + args[0]);
81                 JOptionPane.showMessageDialog(null, "Keyword Invalid " + args[0]);
82             }
83
84 }
```

RPC method is executed by switch statement with (args[0])

```
30     static Scanner sc = new Scanner(System.in);
31
32● public static void main(String[] args) throws InterruptedException {
33
34 //     System.out.println("Retrieve host and port from JmDNS....");
35 //     JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the
36
37     // Receive server information through JmDNS
38     JmDNS jmDNS = JmDNS.create();
39     ServiceInfo[] services = jmDNS.list("_InventoryManagement._tcp.local.");
40     if (services.length == 0) {
41 //         System.out.println("No gRPC server found");
42 //         JOptionPane.showMessageDialog(null, "No gRPC server found");
43 //         return;
44     }
45
46     // Receive host and port through gRPC
47     ServiceInfo serviceInfo = services[0];
48     host = serviceInfo.getHostAddresses()[0];
49     port = serviceInfo.getPort();
50
51 //     System.out.println("Found host and port from JmDNS!!!!");
52 //     JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!!");
53
54
55 //     checking host and port found by JmDNS.
56 //     System.out.println(host);
57 //     System.out.println(port);
58
59     if (args.length == 0) {
60 //         System.out.println("Need one argument to work");
61 //         JOptionPane.showMessageDialog(null, "Need one argument to work");
62 //         return;
63     }
64
65 //     String host = "localhost";
66 //     int port = 50051;
67
68     ManagedChannel channel = ManagedChannelBuilder.forAddress(host, port).usePlaintext().build();
69
70     switch (args[0]) {
71     case "checkItem":
72         checkItem(channel);
73         break;
```

JOptionPane

6. GitHub

From beginning to now, I have committed 40 times for the project.
Detail can be seen below links.

GitHub repo: <https://github.com/Donghyeoklee93/Project-Smart-Warehouse>

Project Commit History :

<https://github.com/Donghyeoklee93/Project-Smart-Warehouse/commits/main>

The screenshot shows the GitHub repository page for 'Donghyeoklee93 / Project-Smart-Warehouse'. The 'Code' tab is selected. The main branch is 'main'. The commit history is displayed in two sections: one for April 23, 2023, and another for April 19, 2023. Each section contains multiple commits by the user 'Donghyeoklee93'. The commits are listed with their messages, dates, verification status (Verified), and commit hash. At the bottom of the page, there is a yellow bar with the text 'GitHub History (From April 18 to now)'.

Donghyeoklee93 / Project-Smart-Warehouse Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main

Commits on Apr 23, 2023

- Build Error handling implementation and Deadline ... Verified 809a47e
- Delete src/main directory Verified f4cd97d
- Delete pom.xml Verified 58be968
- Build Error handling implementation and Deadline ... Verified 612a8d2
- Build Error handling implementation and Deadline ... Verified 4e5481f

Commits on Apr 19, 2023

- modify method name of trackingDelivery service Verified c71008c
- modify method name of trackingDelivery service Verified 3a87c6b
- modify method name of trackingDelivery service Verified bc68b03

Commits on Apr 18, 2023

- Build server in trackingDelivery service Verified 120a5ce
- Build server and client in orderManagement service Verified 9f6160b

Newer Older

GitHub History (From April 18 to now)