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 aleartMessage 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 user 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

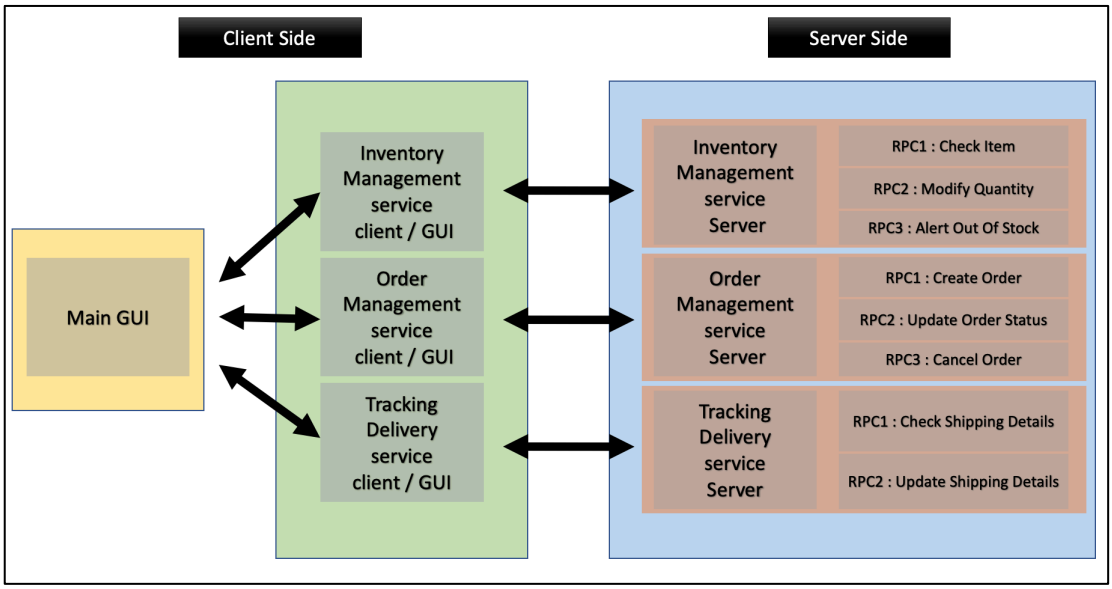
This service is implemented through gRPC. Put simply, there are 3 servers, 3 clients and 1 Main GUI (Integrate 3 services) and 3 clientGUIs. To implement this service, the required steps are like below.

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

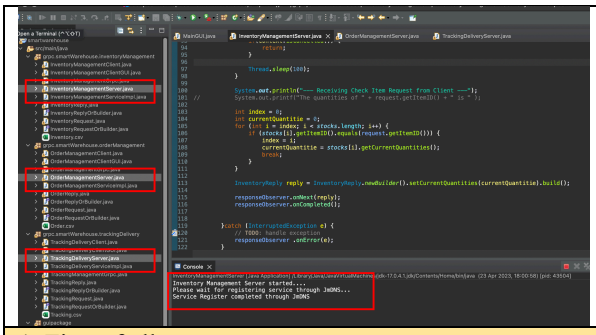
Brief domain structure is like below. User can use this domain by running Main GUI first. This main GUI is linked to 3 separate services' GUI. And by clicking wanted RPC method, user can use RPC methods and interact with database in the server.

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.

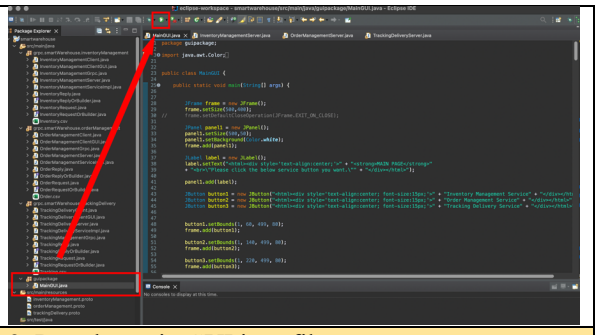


2.3. Service Implementation

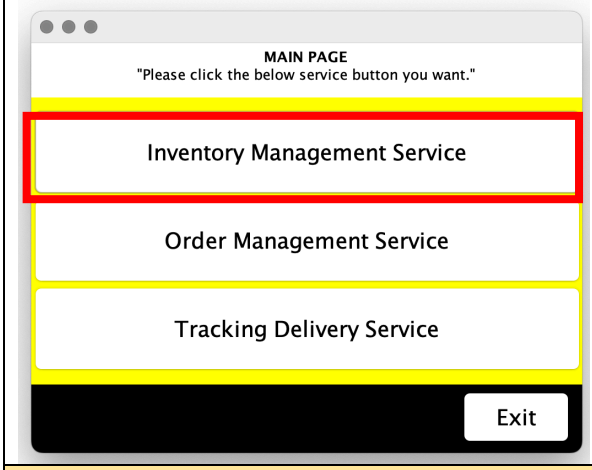
a) InventoryManagementService



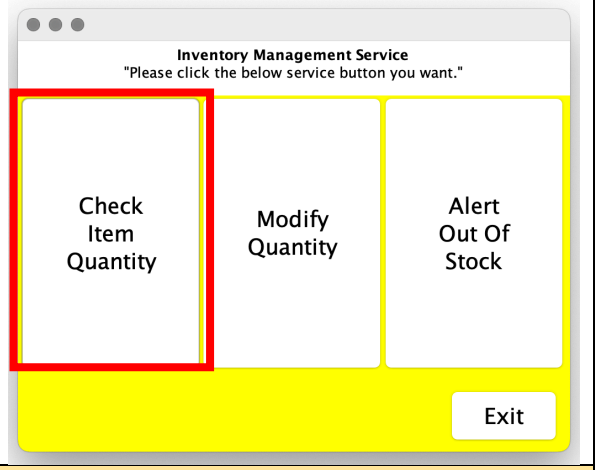
1. First of all, execute servers.
(Through JmDNS, all servers' services are registered)



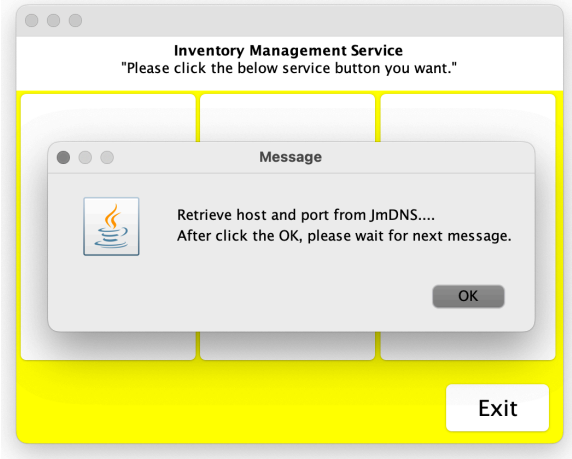
2. Run the main GUI java file.



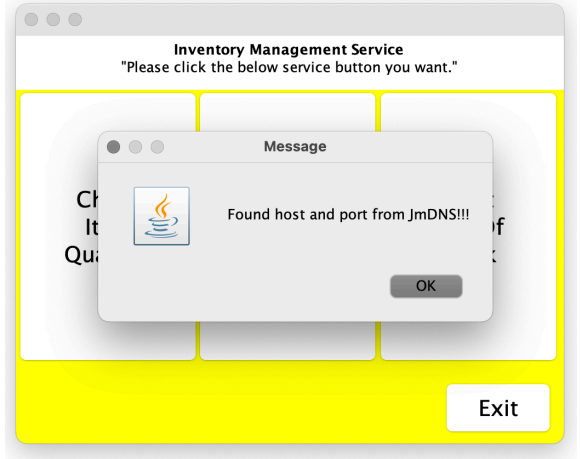
3. Click 'Inventory Management Service'.



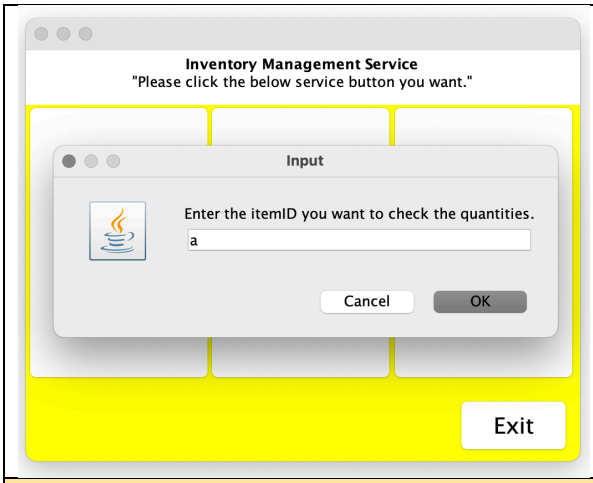
4. Click 'Check Item Quantity'.



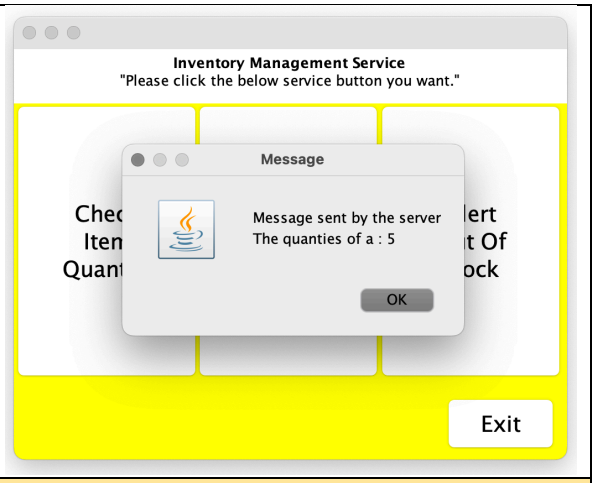
5. Client-side retrieve host and port from JmDNS



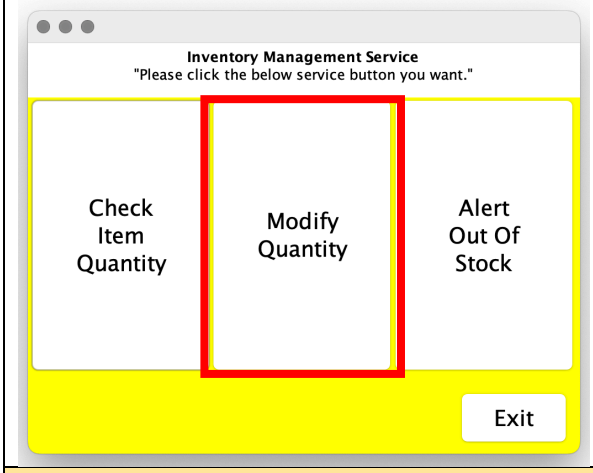
6. In few seconds, Client-side can find host and port form JmDNS.



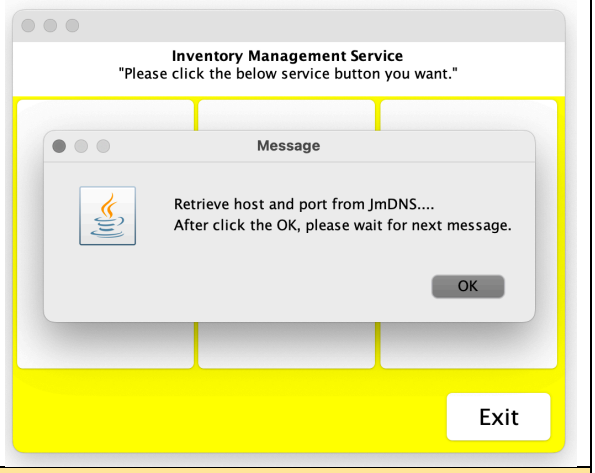
7. Enter itemID user want to check the quantities.



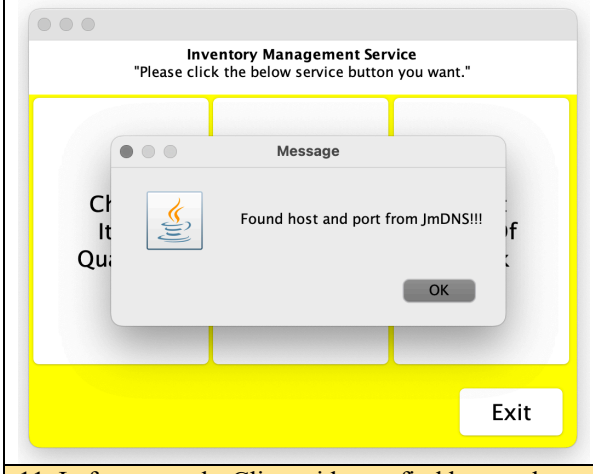
8. From server-side, the message is received.



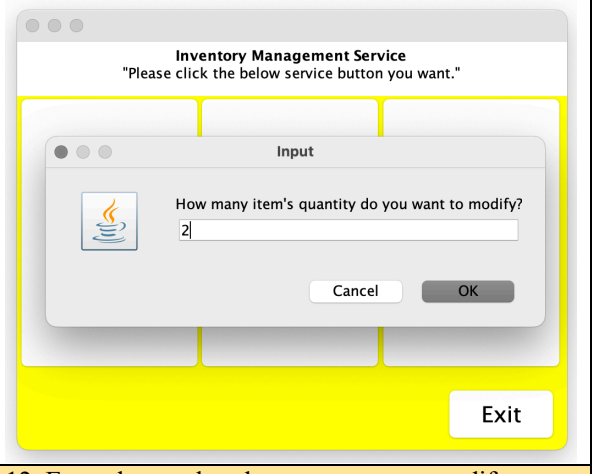
9. Click 'Modify Item Quantity'.



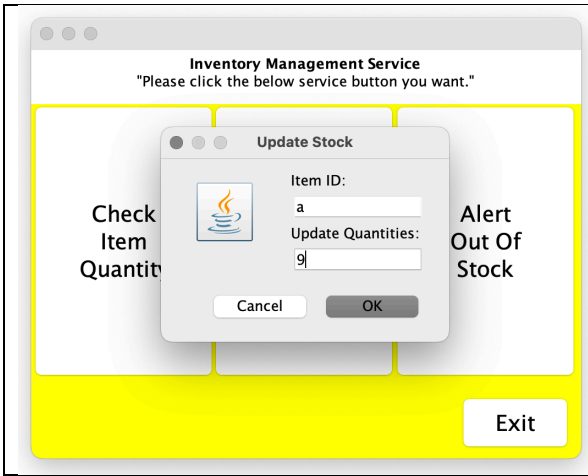
10. Client-side retrieve host and port from JmDNS.



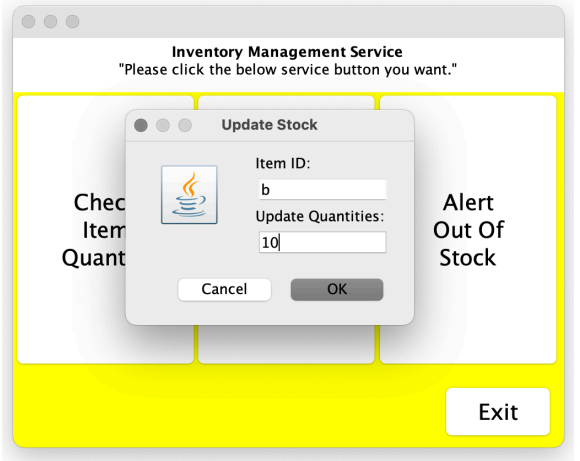
11. In few seconds, Client-side can find host and port form JmDNS.



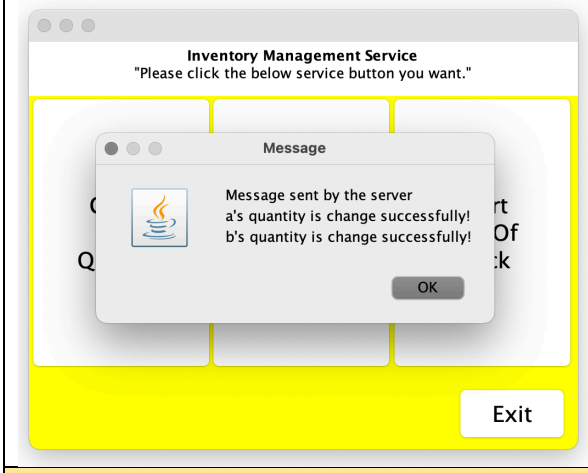
12. Enter the number that user wants to modify quantities.



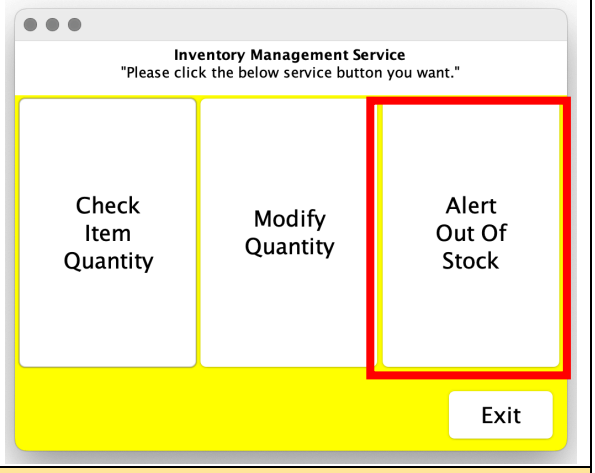
13. Enter the Item ID and Update Quantities (1/2)



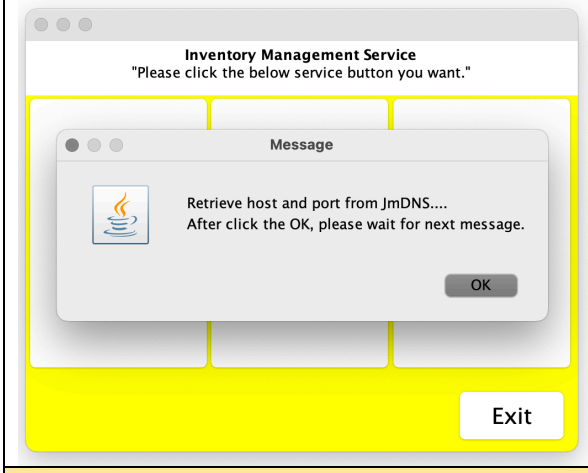
14. Enter the Item ID and Update Quantities (2/2)



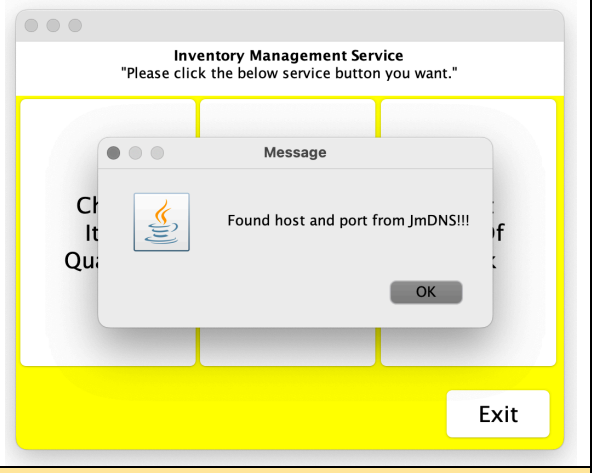
15. From server-side, the message is received.



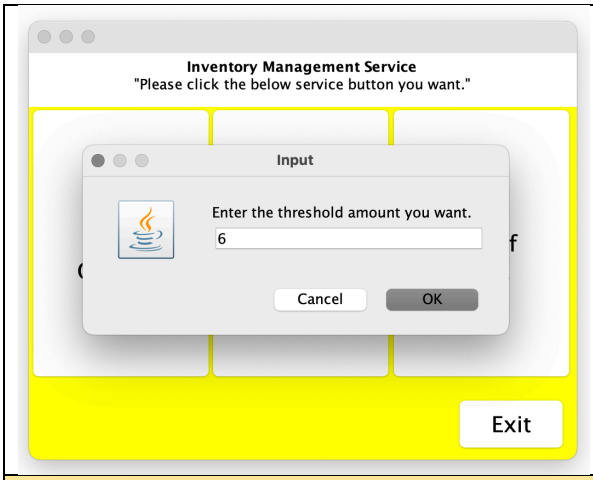
16. Click ‘Alert Out Of Stock’.



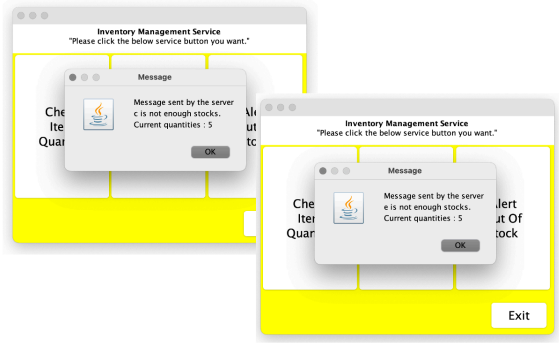
17. Client-side retrieve host and port from JmDNS.



18. In few seconds, Client-side can find host and port form JmDNS.

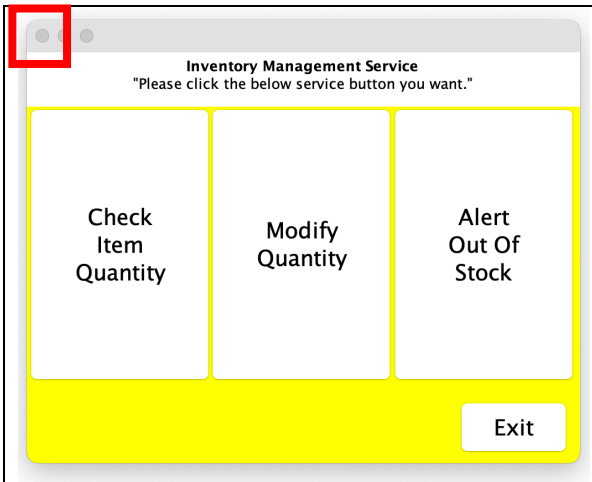


19. Enter the threshold that user wants to check.

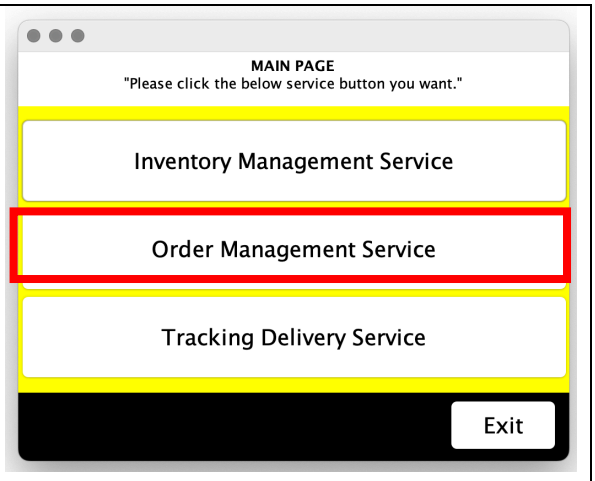


20. From server-side, the message is received.

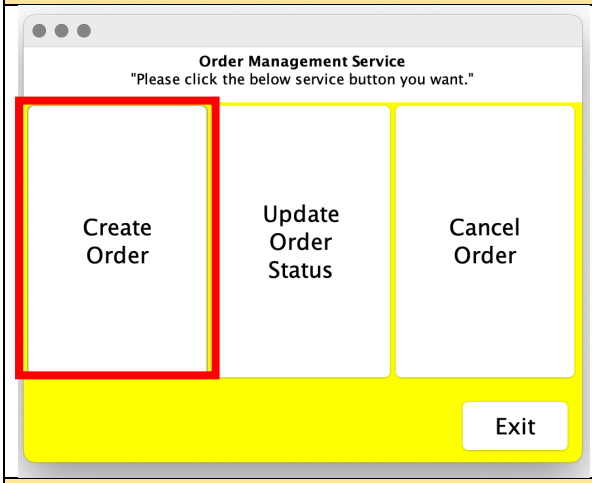
b) OrderManagementService



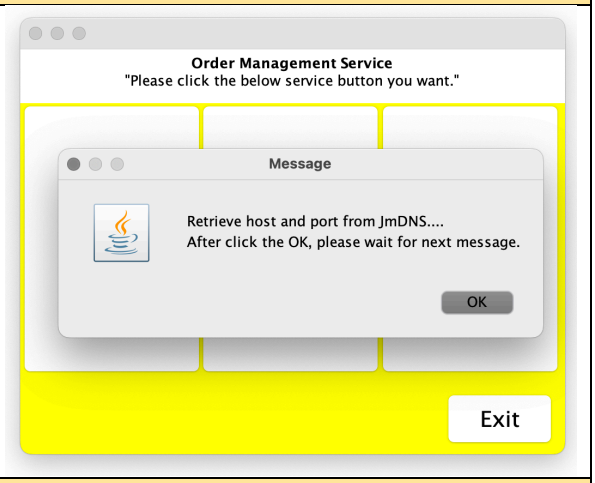
1. Click the 'X' button to return to first page.



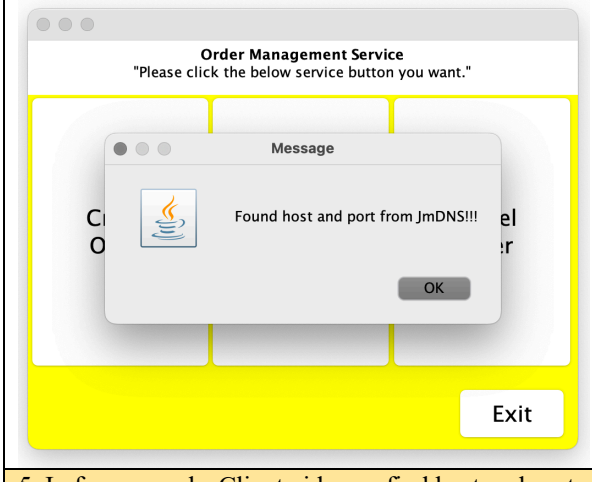
2. Click 'Order Management Service'.



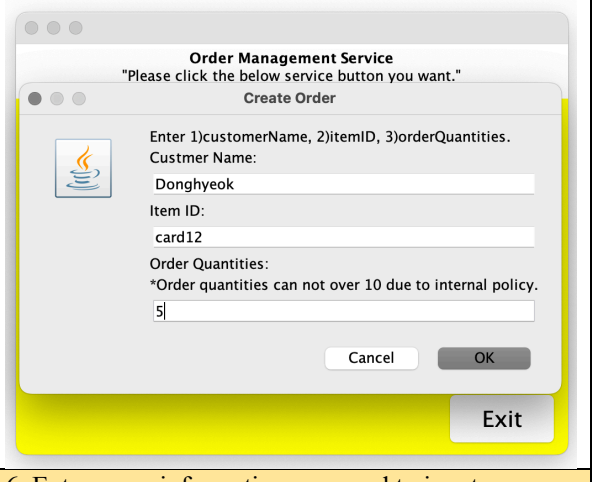
3. Click 'Create Order'.



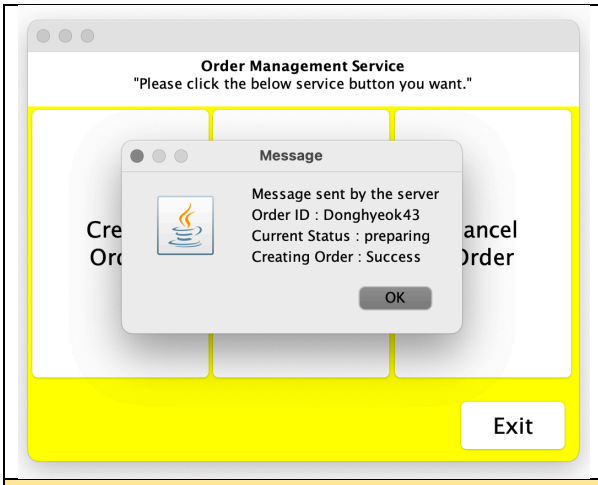
4. Client-side retrieve host and port from JmDNS



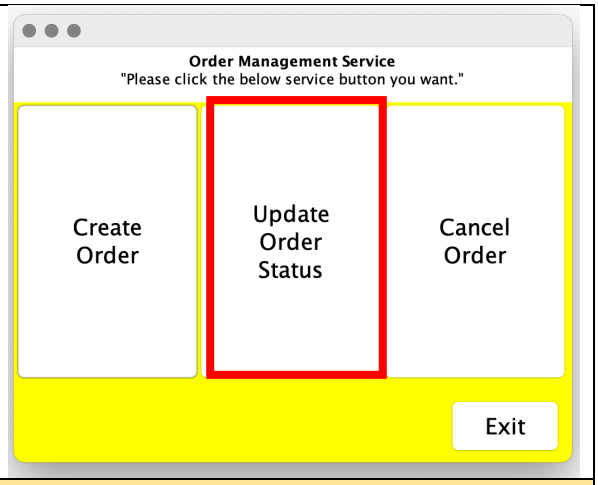
5. In few seconds, Client-side can find host and port form JmDNS.



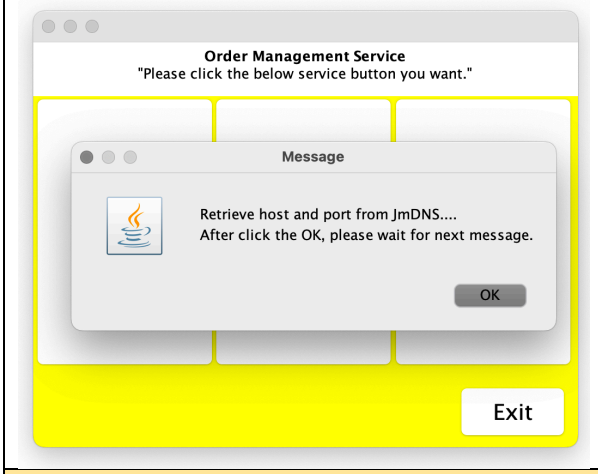
6. Enter some information you need to input.



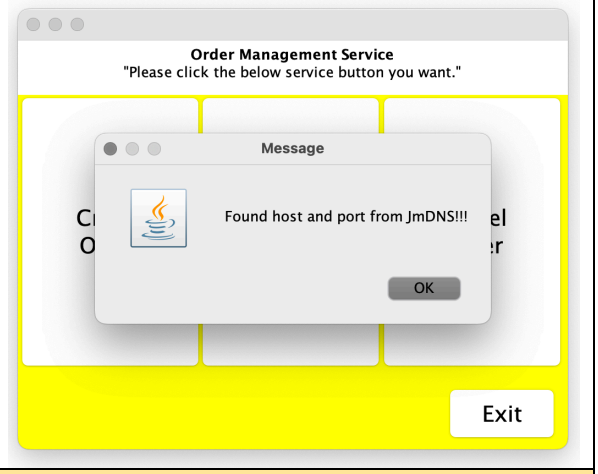
7. From server-side, the message is received.



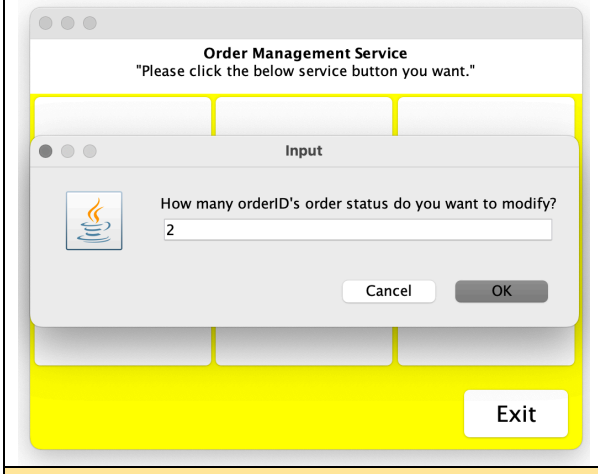
8. Click 'Modify Item Quantity'.



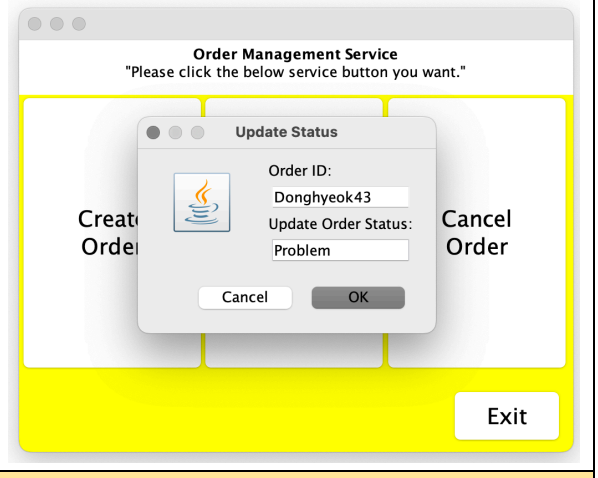
9. Client-side retrieve host and port from JmDNS.



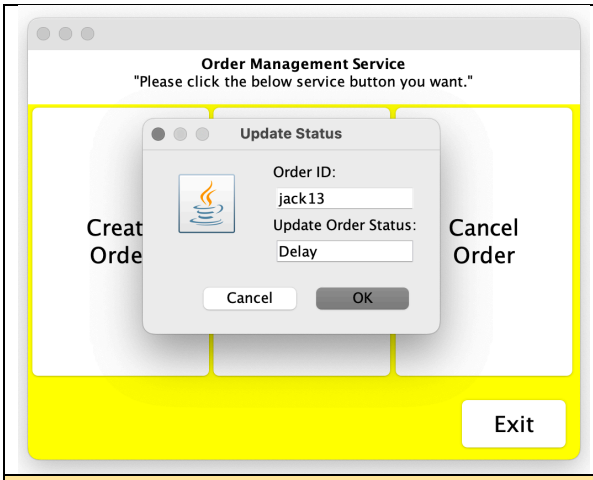
10. In few seconds, Client-side can find host and port form JmDNS.



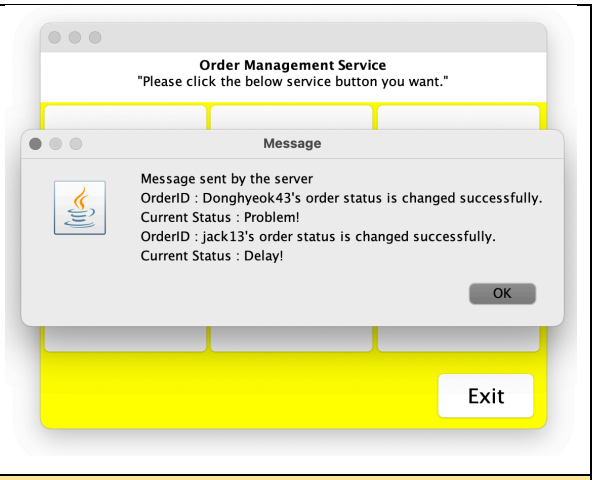
11. Enter the number that user wants to modify order status.



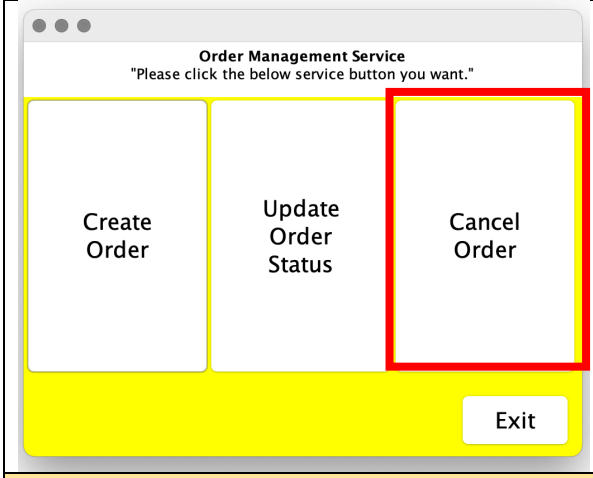
12. Enter some information you need to input (1/2).



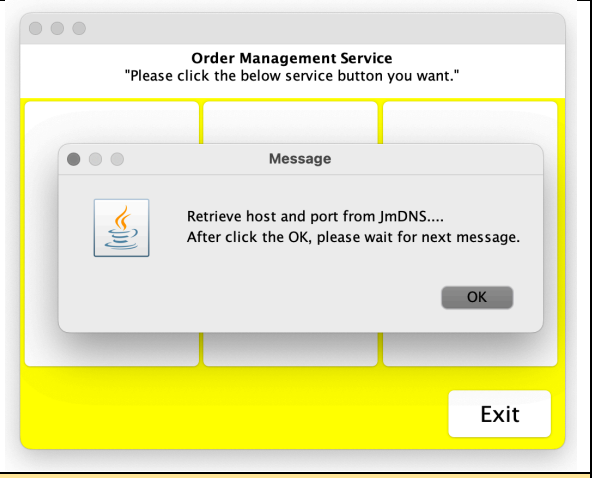
13. Enter some information you need to input (2/2).



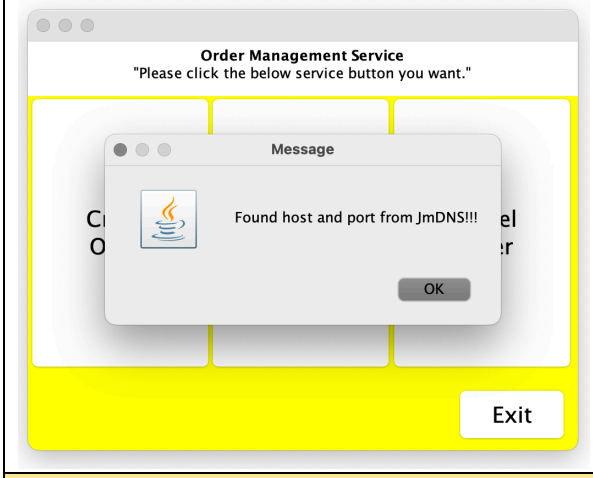
14. From server-side, the message is received.



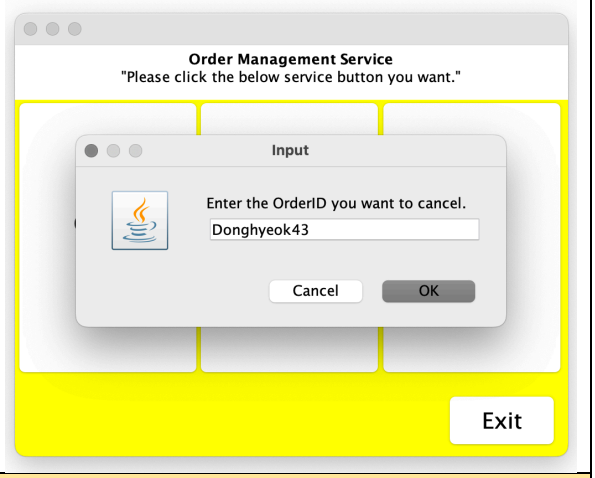
15. Click 'Modify Item Quantity'.



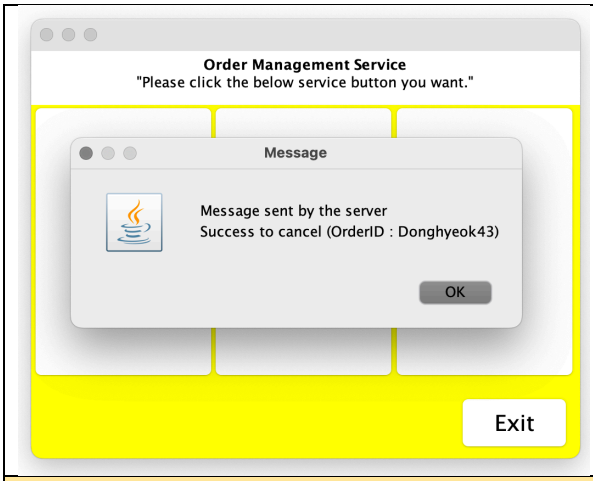
16. Client-side retrieve host and port from JmDNS.



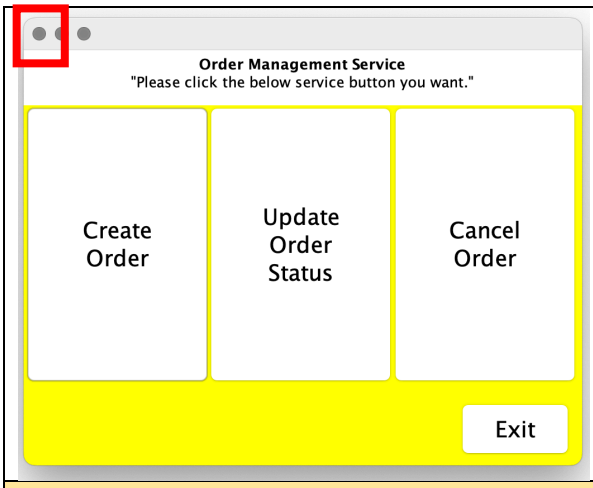
17. In few seconds, Client-side can find host and port form JmDNS.



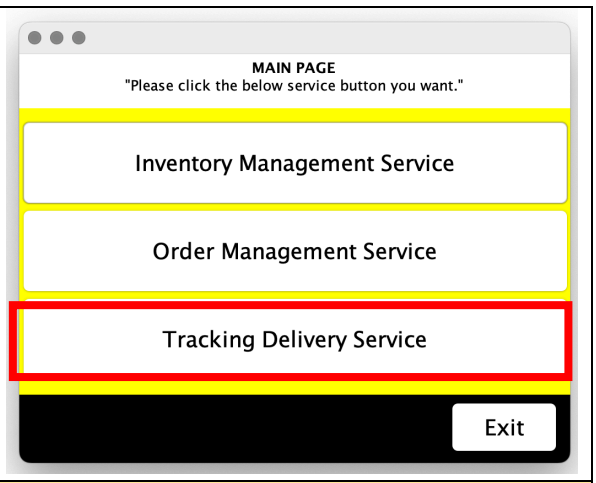
18. Enter OrderID user wants to cancel.

	
19. From server-side, the message is received.	

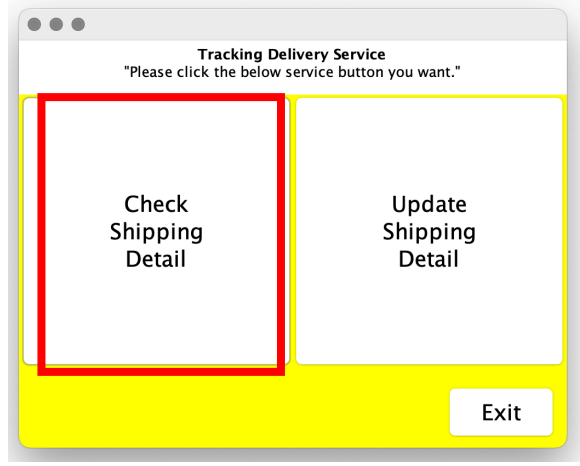
c) OrderManagementService



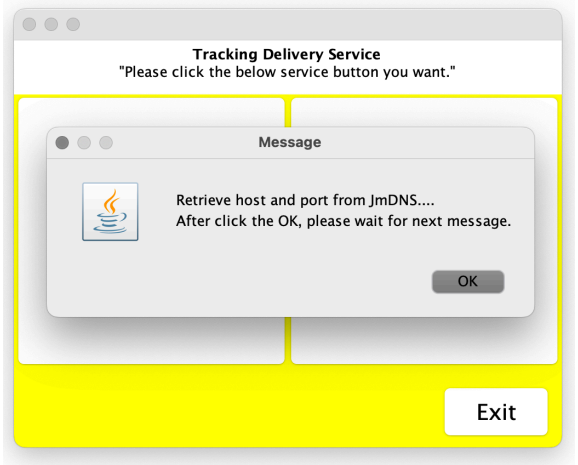
1. Click the 'X' button to return to first page.



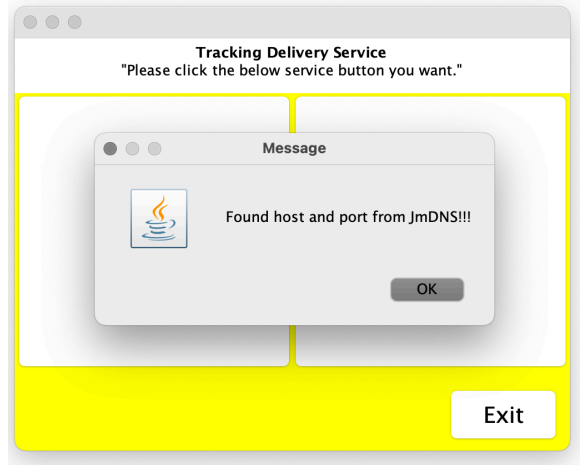
2. Click 'Tracking Delivery Service'.



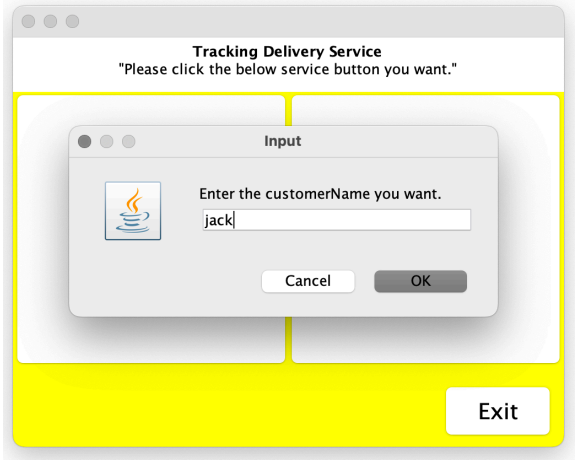
3. Click 'Check Shipping Detail'.



4. Client-side retrieve host and port from JmDNS.

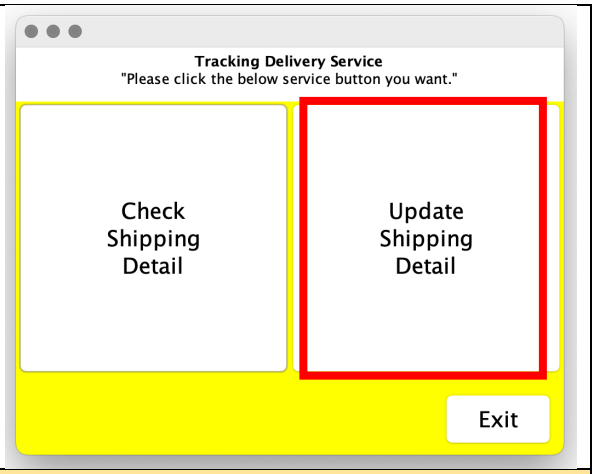


5. In few seconds, Client-side can find host and port form JmDNS.

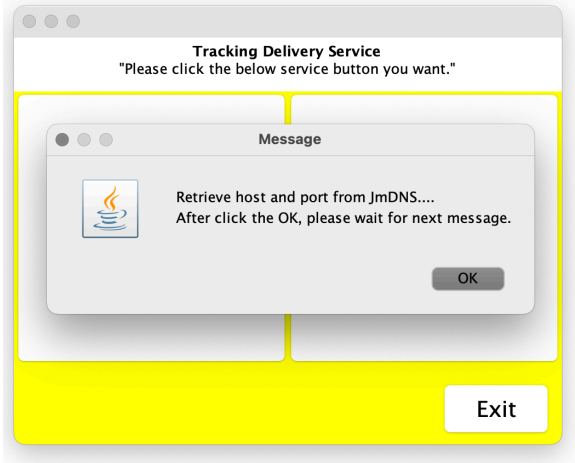


6. Enter the customerName that user wants to check.

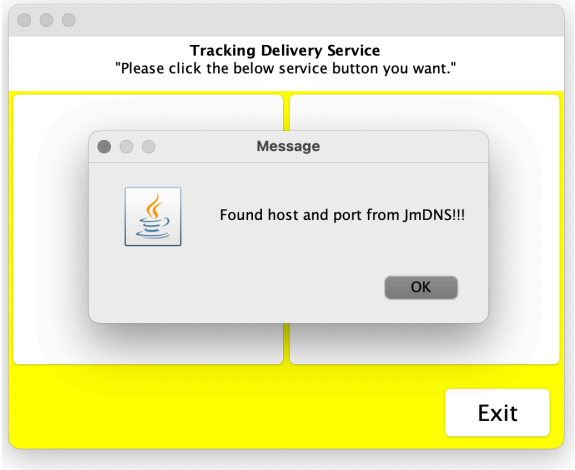
7. From server-side, the message is received.



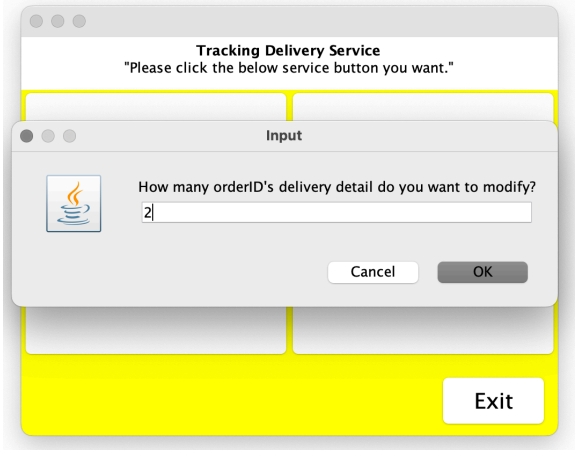
8. Click 'Update Shipping Detail'.



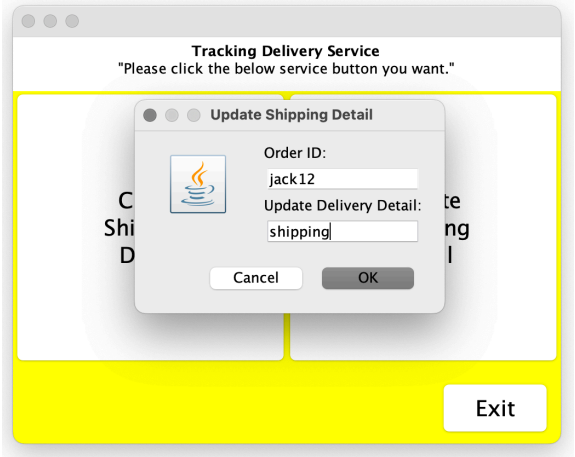
9. Client-side retrieve host and port from JmDNS.



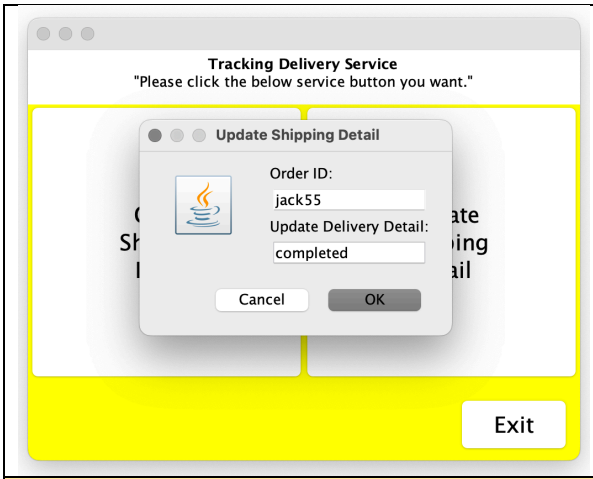
10. In few seconds, Client-side can find host and port form JmDNS.



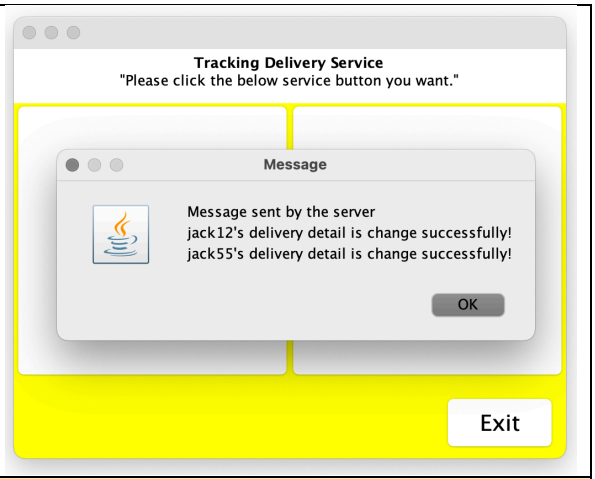
11. Enter the number that user wants to modify delivery detail.



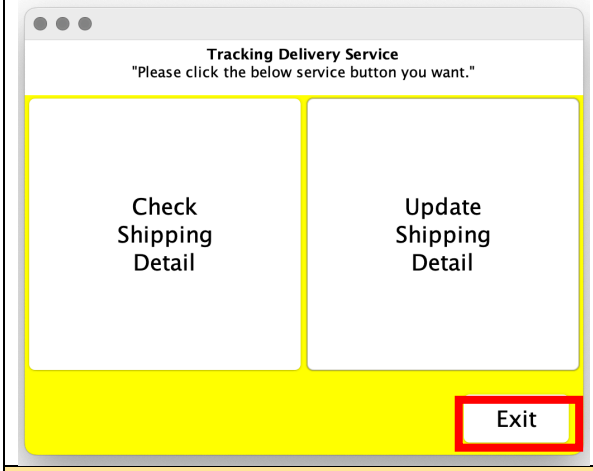
12. Enter some information you need to input (1/2).



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

```
55 //        System.out.println(Arrays.toString(stocks));
56
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          }
78
79      }
```

Code to register service through JmDNS in Inventory Management Server

```
Console
InventoryManagementServer [Java Application] /Library/Java/JavaVirtualMach
Inventory Management Server started....
Please wait for registering service through JmDNS...
```

1. Process to register…

```
Console
InventoryManagementServer [Java Application] /Library/Java/JavaVirtualMachin
Inventory Management Server started....
Please wait for registering service through JmDNS...
Service Register completed through JmDNS
```

2. Finish to register.

```
    public static void main(String[] args) throws InterruptedException, IOException {

//      System.out.println("Retrieve host and port from JmDNS....");
        JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the OK, please wait for next message.");

        // Receive server information through JmDNS
        JmDNS jmdns = JmDNS.create();
        ServiceInfo[] services = jmdns.list("_InventoryManagement._tcp.local.");
        if (services.length == 0) {
//          System.out.println("No gRPC server found");
            JOptionPane.showMessageDialog(null, "No gRPC server found");
            return;
        }

        // Receive host and port through gRPC
        ServiceInfo serviceInfo = services[0];
        host = serviceInfo.getHostAddresses()[0];
        port = serviceInfo.getPort();

//      System.out.println("Found host and port from JmDNS!!!");
        JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!");
```

Client found service through JmDNS

b) OrderManagementService

MainGUI.java | InventoryManagementServer.java | OrderManagementServer.java | TrackingDeliveryServer.java

```
        }
        sc.close(); // closes the scanner

//      System.out.println(orders);

        OrderManagementServer orderManagementServer = new OrderManagementServer();
        int port = 50052;

        try {
            Server server = ServerBuilder.forPort(port).addService(orderManagementServer).build().start();
            System.out.println("Order Management Server started...");

            System.out.println("Please wait for registering service through JmDNS...");

            // Register service through JmDNS
            JmDNS jmdns = JmDNS.create();
            ServiceInfo serviceInfo = ServiceInfo.create("_OrderManagement._tcp.local.", "grpcServer2", port, "server2");
            jmdns.registerService(serviceInfo);

            System.out.println("Service Register completed through JmDNS");


            server.awaitTermination();
        } catch (IOException | InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

// RPC Method 1 : Create Order (Unary RPC)
```

Code to register service through JmDNS in Order Management Server

smartwarehouse/src/main/java/guipackage/MainGUI.java

```
Order Management Server started...
Please wait for registering service through JmDNS...
```

1. Process to register…

Console

```
OrderManagementServer [Java Application] /Library/Java/JavaVirtualMachine
Order Management Server started...
Please wait for registering service through JmDNS...
Service Register completed through JmDNS
```

2. Finish to register.

```
//      System.out.println("Retrieve host and port from JmDNS....");
        JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the OK, please wait for next message.");


        // Receive server information through JmDNS
        JmDNS jmdns = JmDNS.create();
        ServiceInfo[] services = jmdns.list("_OrderManagement._tcp.local.");
        if (services.length == 0) {
//          System.out.println("No gRPC server found");
            JOptionPane.showMessageDialog(null, "No gRPC server found");
            return;
        }

        // Receive host and port through gRPC
        ServiceInfo serviceInfo = services[0];
        host = serviceInfo.getHostAddresses()[0];
        port = serviceInfo.getPort();

//      System.out.println("Found host and port from JmDNS!!!");
        JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!");
```

Client found service through JmDNS

c) TrackingDeliveryService

```java
TrackingDeliveryServer trackingDeliveryServer = new TrackingDeliveryServer();
int port = 50053;

try {
    Server server = ServerBuilder.forPort(port).addService(trackingDeliveryServer).build().start();
    System.out.println("Tracking Delievery Server started...");

    System.out.println("Please wait for registering service through JmDNS...");

    // Register service through JmDNS
    JmDNS jmdns = JmDNS.create();
    ServiceInfo serviceInfo = ServiceInfo.create("_TrackingDelivery._tcp.local.", "grpcServer3", port, "server3");
    jmdns.registerService(serviceInfo);

    System.out.println("Service Register completed through JmDNS");


    server.awaitTermination();
} catch (IOException | InterruptedException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Code to register service through JmDNS in Tracking Delivery Server

```
Console
TrackingDeliveryServer [Java Application] /Library/Java/JavaVirtualMachines/jd
Tracking Delievery Server started...
Please wait for registering service through JmDNS...
```

1. Process to register…

```
Console
TrackingDeliveryServer [Java Application] /Library/Java/JavaVirtualMachines/
Tracking Delievery Server started...
Please wait for registering service through JmDNS...
Service Register completed through JmDNS
```

2. Finish to register.

```java
public static void main(String[] args) throws InterruptedException, IOException {

//      System.out.println("Retrieve host and port from JmDNS....");
        JOptionPane.showMessageDialog(null, "Retrieve host and port from JmDNS.... " + "\nAfter click the OK, please wait for next message.");


        // Receive server information through JmDNS
        JmDNS jmdns = JmDNS.create();
        ServiceInfo[] services = jmdns.list("_TrackingDelivery._tcp.local.");
        if (services.length == 0) {
//            System.out.println("No gRPC server found");
            JOptionPane.showMessageDialog(null, "No gRPC server found");
            return;
        }

        // Receive host and port through gRPC
        ServiceInfo serviceInfo = services[0];
        host = serviceInfo.getHostAddresses()[0];
        port = serviceInfo.getPort();

//      System.out.println("Found host and port from JmDNS!!!");
        JOptionPane.showMessageDialog(null, "Found host and port from JmDNS!!!");
```

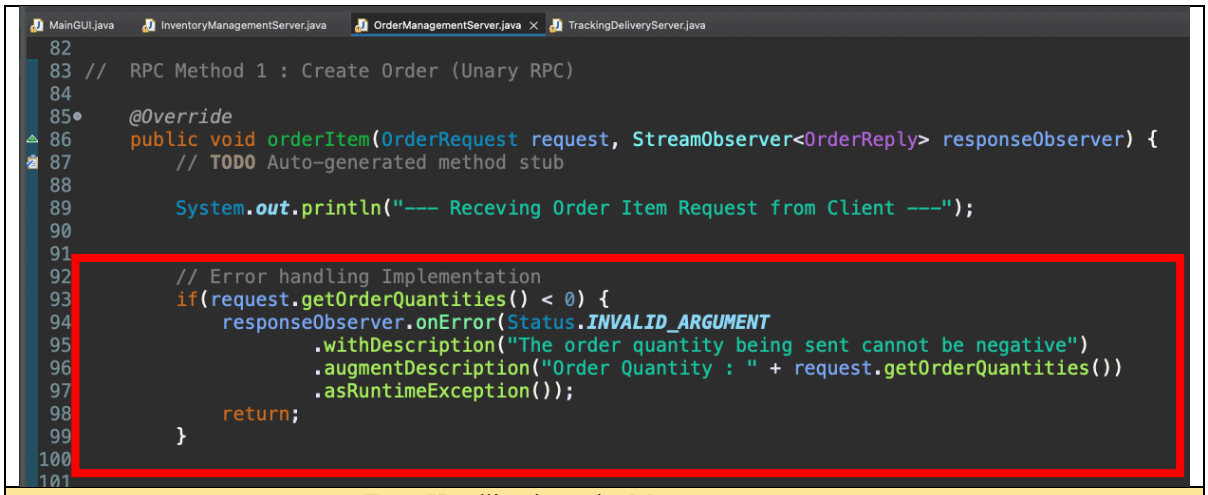
Client found service through JmDNS

4. Error Handling & Advanced Features

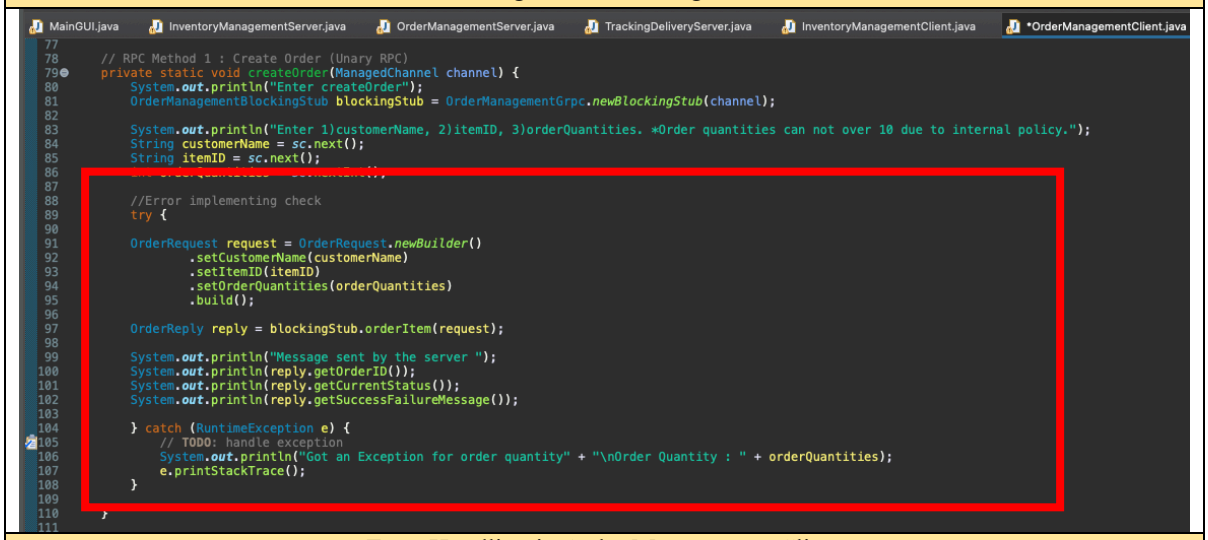
4.1. Error Handling

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.

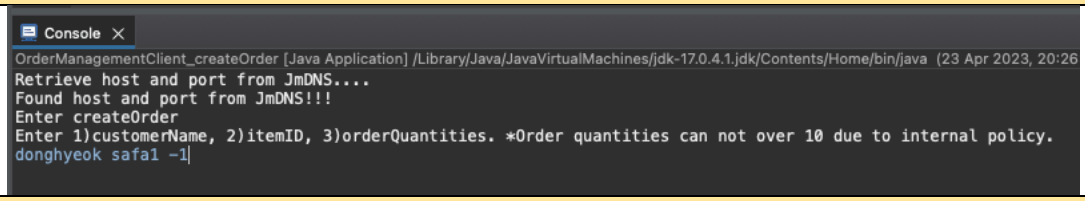
- **Input negative value for 'orderQuantities' in 'Create order' method of Order Management Service.**



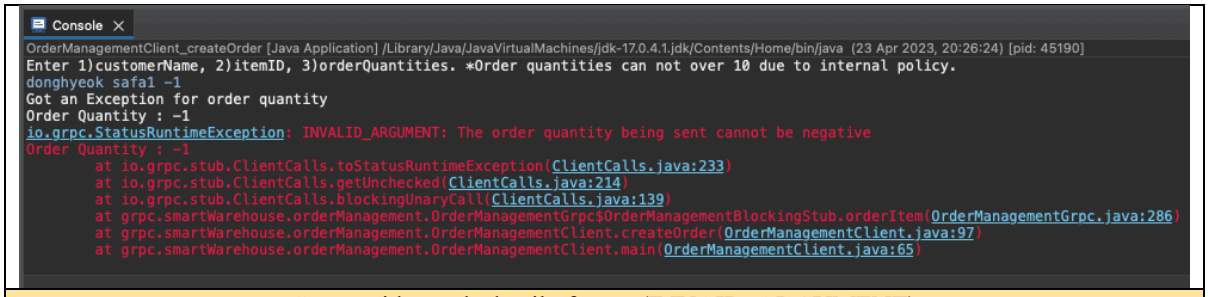
Error Handling in Order Management Server



Error Handling in Order Management Client



Enter '-1' for orderQuantities (Inproper input)

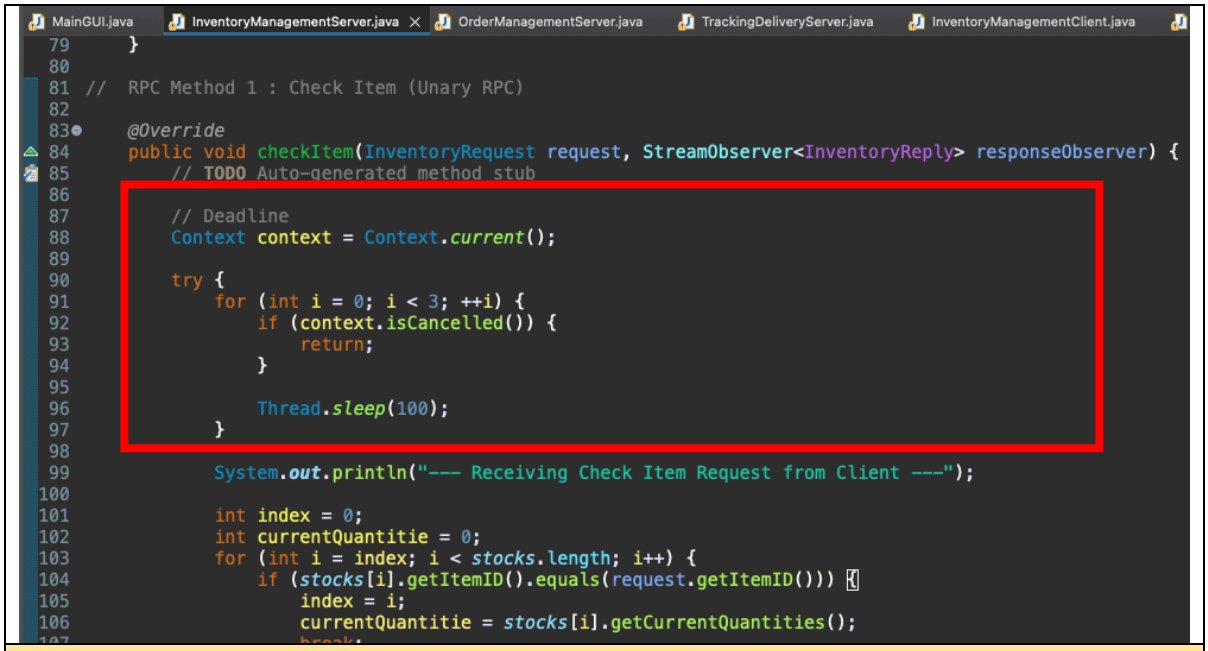


Server-side sends detail of error (INVAID_ARGUMENT)

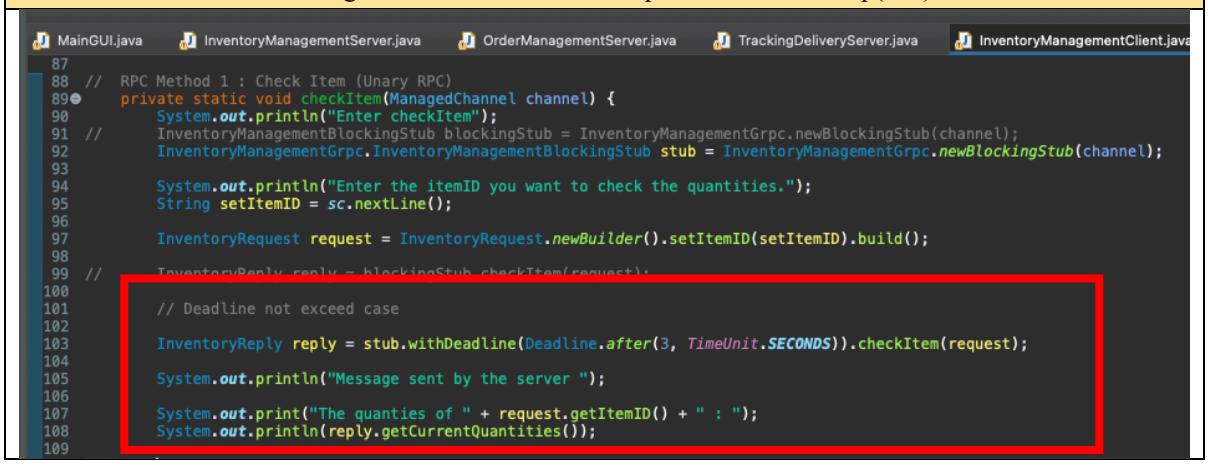
4.2. Deadlines

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.

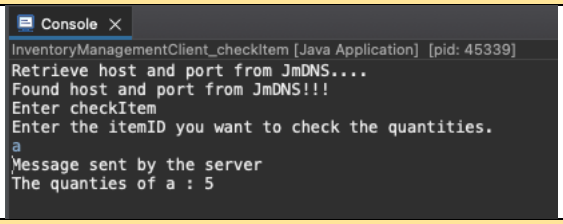
- **Deadline in Inventory Management Server and Client-side**



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

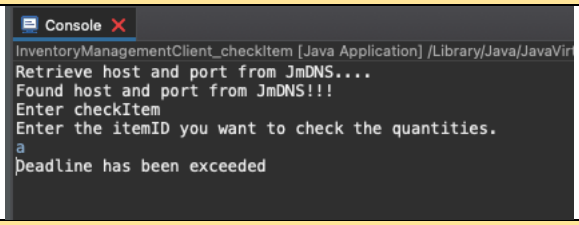


In client-side, deadline.after is 3 seconds, so it's much than Thread sleep time in server.



So, server sends message normally.

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



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
41          panel1.add(label);
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

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

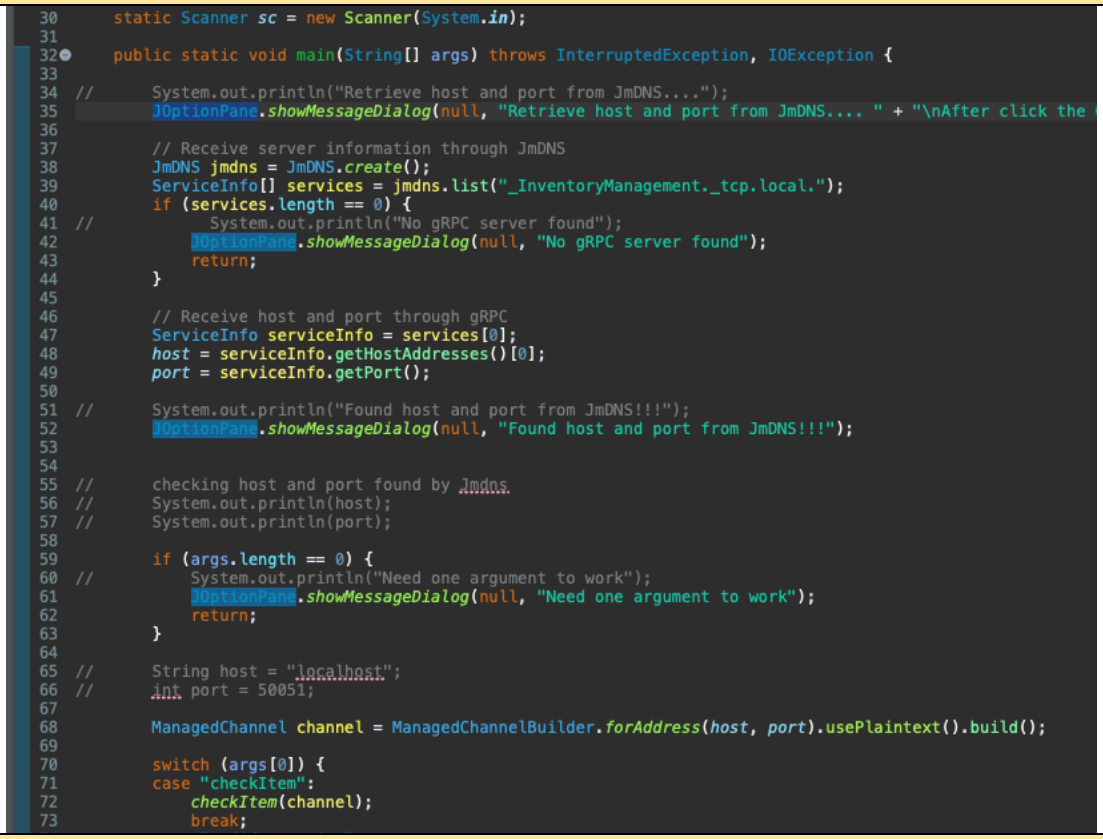
addActionLisener

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
70          switch (args[0]) {
71          case "checkItem":
72              checkItem(channel);
73              break;
74          case "modifyQuantity":
75              modifyQuantity(channel);
76              break;
77          case "alertOutOfStock":
78              alertOutOfStock(channel);
79              break;
80          default:
81  //          System.out.println("Keyword Invalid " + args[0]);
82              JOptionPane.showMessageDialog(null, "Keyword Invalid " + args[0]);
83          }
84
85      }
```

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, IOException {
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;
```

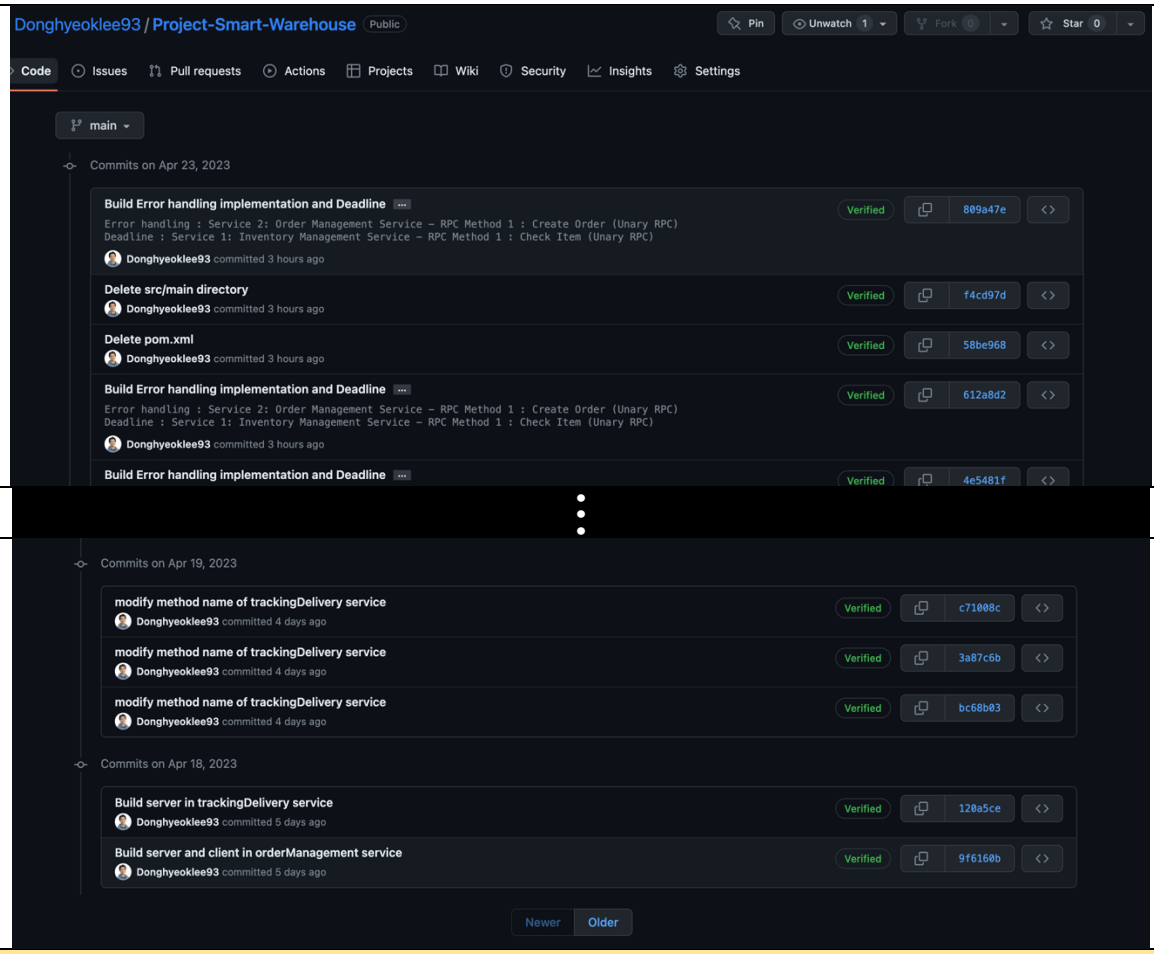
JOptionPane

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



GitHub History (From April 18 to now)