

Distributed Algorithms and Systems Assessed Coursework

Lim Bing Shun
2228131L

1 Introduction

The specification of this assessed exercise is to design and implement an online Auction System using Java RMI. In this report, I will discuss on the overall design, design goals, design requirements, potential extensions, performance of the system and test case on how I have tested the program.

2 Overall Design of the Auction System

The overall design of the Auction System is to have a Client sending intended instruction to the Server and Server will do logical computation and return the result to the Client. There are 3 server classes, 3 client classes, one object class and one Constant class. I have also included a test class to be used for performance testing.

2.1 Client

A platform that users use to create auction item, bid an auction item, and retrieve information of all auction items. Server will also notify the client relevant information. It also check the server for its alive status and terminates itself if the server is down. There are 3 classes in this section.

- Client.java: This is used to provide general information for the user. When the user uses the Client function to query for an Auction item, the Client will send a request from the Server, and wait for the Server to reply. After which, the Client will then display the data from the server to the Client in readable format. It also instantiate the Client object for the Server and the connection to the Server.
- AuctionClientInterface.java: This is used for creating a Client Instance for server callback
- AuctionClientImpl.java: This is used to implement the Client interface to allow server to perform various function back to the client or to get information from the client instances.

2.2 Server

The server is used to start the connection between Client and the information of shared object, validate and processing of the user input and store information of the object, perform the task that the user has queried, notify the user if an error has occurred while working on their request or to let the user knows any information in regards to the auction, able to end an auction to prevent the other users from further bidding it, able to save and restore state of the server, able to handle concurrent access from multiple client connections, able to know which client are online and to block multiple of same client id login at the same time. There are 3 classes in this section.

- Server.java: It initiates the Server interface to be ready for client to connect. It works as a bridge to connect the server and the client.
- AuctionServerInterface.java: It contains all the method that a client can call to perform various functions that is available.
- AuctionServerImpl.java: It contains all the logical computation and functions. This is bind with AuctionServerInterface so that the client can return the result that the system intends to.

2.3 AuctionObject.java

A class to store information of Auction object. It is used by the Server to store information and retrieve information from the Client request.

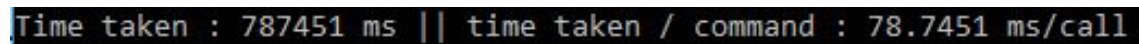
2.4 Constant.java

This is to store default value in the system.

3 Does the solution works as advertised

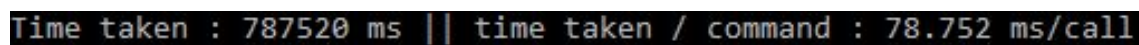
The designed solution works as per specification required. The system is tested both in Windows and Linux version with no intended error. A set of task is used to test the system. The table of task and its screenshot is shown in the next section. I have also included a set of instruction in the README file attached to the submission zip to compile the java file, to start the server and to start the client.

4 Performance of the system



```
Time taken : 787451 ms || time taken / command : 78.7451 ms/call
```

Figure 1: Time taken for 10k functional call on Client01



```
Time taken : 787520 ms || time taken / command : 78.752 ms/call
```

Figure 2: Time taken for 10k functional call on Client02

The system is being tested with two user creating an auction item for 10000 times. The result is shown below how the system performance of the system. There are two reasons where the system performance is being compromised. First is when the user creates an auction item, it save the state of the system. Every state save is to write a new system state to the server storage, thus the performance for this depends on the writing speed of the storage. The other factor that contributes to the performance is that when the client creates an auction item and save the state, the functions are locked by “synchronized”, which means at one point of the time, only one user is able to access to function and the other users have to wait for that function. Thus this greatly reduce the performance as to the performance of a system without having the functions locked. The reason why I have implemented synchronized to this function is that, if multiple user is to commit to the system, it will not corrupt the state as this function is performing as though it is serializable. Thus there should not be any corruption of the data.

5 Discussion of extensions to improve fault tolerance and server availability

5.1 Fault tolerance

The possible extension would be, when a server is down while client has execute the command, instead of cutting the client out of the server, it should pause what the client is doing (unless client explicitly disconnect themselves), until when the server to come alive, the client interface should re-execute the command. Following “at-least-once” semantic invocation.

5.2 Availability of the server

The server should create redundancy for the server. As such if one server is down, the other two server is still alive and ready to receive client’s input. When the server that is down comes

back alive, the other two server will then update that server. This is to help the server to reduce downtime of its service to the client.

6 Testing of the program

Task no	Task	Working?
1	Starting of server	Yes
2	Starting of client	Yes
3	Client creates bid	Yes
4	Client bids for an item	Yes
5	Client listing out bids	Yes
6	Notify owner after a bid	Yes
7	Notify bidders after a bid	Yes
8	Save state	Yes
9	Restore state	Yes
10	Check server availability	Yes
11	Check connected client	Yes

Table 1: Task table.

6.1 Task 1: Starting of Server

```
C:\Users\Vince\Desktop\Final>java Server
[2016-11-24 23:29:53.625] Unable to restore state
[2016-11-24 23:29:53.63] Server has started
```

Before the server starts, it will restore the previous state of the server. If the server is relatively new without any previous server state, the system will not be able to restore any state, thus it will prompt out “Unable to restore state” on the server.

```
C:\Users\Vince\Desktop\Final>java Server
Server has restore state successfully
Previous save state has been restored
[2016-11-24 22:30:39.537] Server has started
```

Whereas if there is a previous server state, the server will restore its previous state.

6.2 Task 2: Starting of client

```
C:\Users\Vince\Desktop\Final>java Client
Welcome to Auction System.
Before you begin, please enter your username:
TestingACCOUNT01
Hello TestingACCOUNT01
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program
```

The client is able to start up when the server is online.

```
[2016-11-24 22:46:00.537] Testing Account 01 has triggered checkUser
[2016-11-24 22:46:00.538] Testing Account 01 is now connected
```

On the server side, it will notify the administrator that the user is now connected to the server. The checkUser function is to prevent duplicate user from coming online to the Auction System.

6.3 Task 3: Client creates bid

```
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program

1
Item name :
Testing Item 01
Minimum bid value :
100
Duration for auction (in seconds) :
600
Notification : Successfully listed your item with item ID : 0
```

When an item got listed, the system will notify the item owner that the item has successfully listed in the system, returning the item's ID.

6.4 Task 4: Client bids for an item

```
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program

2
Please enter the Bid ID of the item:
0
You cannot bid your own item.
```

When the owner tries to bid his own item, it will be stop by the system.

```
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program

2
Please enter the Bid ID of the item:
0
Please enter your bid amount :
1
Bid amount is lower than the start bid or bid has already ended. Please try again.
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program

2
Please enter the Bid ID of the item:
0
Please enter your bid amount :
101
Notification : Bid Accepted
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program
```

If the user enter a bid amount that is lower or already ended, the system will prompt the user with an error message. If the user manages to bid an item, the server will notify the user that the bid is accepted by the system.

6.5 Task 5: Client listing out bids

```
What would you like to do ?
1. Create a bidding item
2. Bid for an item
3. View all item that has listed
4. Exit the program

3
Owner: TestingACCOUNT01
ID: 0
Item Name: Testing Item 01
Starting bid: 100.0
No of Bidders: 1
Closing Date: 2016-11-24 23:41:44.274
Auction Status: Available
```

Client is able to list out the available list of auctions. Those auction item that has passed auction date will be remove within a minute (for testing purposes). Client will not be able to bid for item that had the auction status closed (Not Available will be displayed in the system).

6.6 Task 6: Notify owner after a bid

```
Notification : Auction of Testing Item 01 (ItemID: 0) has ended.
Notification : Your item Testing Item 01 (ItemID: 0) has been bought by TestingACCOUNT02 @ $20000.0
```

Item owner will be notified that the item auction ends and the winner of the auction.

6.7 Task 7: Notify bidders after a bid

```
Notification : You have won the auction of the Testing Item 01 item ID : 0
```

The bidder who won the auction will have notification stating that he won the auction.

```
Notification : Winner of Testing Item 01 (itemID: 0) is TestingACCOUNT02 with the price of 20000.0
```

The bidder who did not win the auction will be notified on who (which user) won the auction item that they have bid and how much they have bid.

6.8 Task 8: Save state

```
[2016-11-24 23:31:44.28] TestingACCOUNT01 has triggered createBidObject
[2016-11-24 23:31:44.301] Successfully save the system state
```

```
[2016-11-24 23:33:10.479] TestingACCOUNT02 has triggered bidItem
101.0
0.0
TestingACCOUNT02
[2016-11-24 23:33:10.488] Successfully save the system state
```

The server will save the state if the client has made any changes to the system (create bid, bid an item when the auction listing has expired).

6.9 Task 9: Restore state

As mention in Task 1, the state will be able to restore when the server start up, if the restore file is not present, it will notify the server administrator that the server is not able to store previous state.

6.10 Task 10: Check server availability

```
Unable to reach server ... Server might be downed. Please try again later..
Client is now exiting...
```

When the server is down, the client program will notify the user that it is not able to reach the server and thus exit the program after a few seconds.

6.11 Task 11: Check connected client

```
[2016-11-24 23:36:48.094] X is now connected
[2016-11-24 23:36:51.619] X has triggered bidItem
12222.0
101.0
X
[2016-11-24 23:36:51.628] Successfully save the system state
[2016-11-24 23:37:14.433] X has triggered viewItem
[2016-11-24 23:37:36.02] TestingACCOUNT02 has triggered bidItem
20000.0
12222.0
TestingACCOUNT02
[2016-11-24 23:37:36.029] Successfully save the system state
End auction of Testing Item 01 with ID: 0
[2016-11-24 23:43:33.918] X is now disconnected
```

Server keeps a log on which user is connected and disconnected from the system.