Programmer’s Point of View

My code starts by creating a Server class that extends Observable. Within the class, I have a global instance of an Auction, a variable that holds the port number which is configured during launch with the server, and arraylists that will hold the contents of a provided item list (in my case, a text file). As the launch of the Server.java file starts, the main method of the Server class calls on the viewController class’s start method which creates a gui for the server and sets up the network. During the set up network method, the server waits for a client to make a connection to the socket. From here, the method adds the user (which is the name the client was given) to the global users on the server. The method then reads the text file in order to update the arraylists of items on the server and sends the items to the client. If someone other than the first user to connect joins the server, the set up network method will add this user to the global users on the server and use the already existing arraylist of items on the server to send the newly connected client an updated item list (this includes any newly appended items as well as any newly placed bids on items by previous clients).

As far as the code on the Client side, I have a Client class that extends the Application class. Within this class, I have many static variables and 4 primary methods. These methods handle the different stages/phases of the client as it is launched and run. The first method handles the connection to the server. Once connected to the server, the second method handles the login and creation of a user. After user creation, the third method obtains the items being sold on the server. While each method has an accompanying gui for the user to interact with, the final method creates a separate window that is created when the user clicks on an item of their choice so that the user can see the details of an item such as the if the item is still being sold, the current highest bid, the current winner, and an item description. The gui of the third method has the bulk of the features for the user to interact with. Here, the user has a history log of bids made in real time (including themselves), the ability to place a bid, and a list of items being sold on the server. If the user places a bid that is invalid, they will be informed by a pop up of why their bid is invalid. Within the Client class, I have an IncomingReader class that handles all input from the Server after client initialization.

User’s Point of View

From the user’s standpoint, the client will prompt the user to enter the ip address and port being used by the host’s server. If the host’s ip address happens to be the user then there is an interfaced ‘My IP’ button which will autofill your ip address. Once the ip address and port number text boxes have been filled, and assuming the user has entered the correct ip and port number of an ongoing server, the user can click the connect button. If there isn't an ongoing server with the specified ip and port number, the client will inform the user through a popup that this is the case. From here, the client interface will prompt the user to enter a username and password of their choice into text boxes. Once the user has chosen these, they can click the login button. The client now shows the user the ongoing auction being held by the server. The client features a log history of placed bids, a textbox for the user to enter a bid on an item of their choice, and an item list on the right side of the client. If the user clicks on an item of the item list, a pop will appear which gives the user details on the selected item such as the if the item is still being sold, the current highest bid, the current winner, and an item description.

Optional features added include:

- Set a minimum starting price > 0 for every item.
- Set the duration for the auction for each item separately.
- Set a high limit that is a ‘Buy It Now’ price. When a customer bids that amount, he/she gets it right away.
- Nice GUI in general.
- Using the Observable class and Observer interface.