**CS 471 Project 3**

**General Information**

The project should be run on the Mason server (technically, either should work). Compile each of the three files AuctionServer.java, Buyer.java and Seller.java. All other java files will be compiled as well.

**High-Level Explanation**

Both clients (Buyer & Seller) function similarly. They will attempt to connect to the AuctionServer on the given hostname and port number. If successful, it will input and output readers, start a thread that is used to receive responses from the server and initiate a loop to send messages to the server. The thread is used to receive messages from the server. This seperation will allow the client to receive messages from the server even when the client is idle. Once the client receives the appropriate response from the server, it will terminate the thread. This will cause the main loop in main() to terminate and close the connection to the server.

The server will setup a socket, bind the hardcoded address to the socket and listen for connections. I originally wrote this to listen to one port and when I made changes to add the second I forgot that the accept command for sockets blocks until it receives a signal. I found two ways to work through this. Due to time constraints, I set the sockets to timeout every .25 seconds. This causes an exception that I have conveniently ignored. The other option was to use a Selector. While this method is preferable, time constraints limited my ability to implement it. As such, I have stuck with the slightly messier timeout method (I guess this is more of a workaround). Once a client successfully connects, a thread is created and added to an array. The client thread contains the majority of the code. In here, input from the client is processed and output returned. Each thread receives a reference to the thread array, as well as the message queue and auction item list. Commands are processed depending on client permissions and responses are sent to the user as well as any related users. These related users do not have to be connected as their notification is saved to a message queue and delivered to that user the next time they successfully connect.

**Data Structures**

**Auction Server** - Runs primarily in a while loop. In this loop, the server listens on both ports for a client connection attempt. If successful, the server spawns a thread and assigns it as an element in an array of threads (see Client Thread) . The thread also receives the thread array, message queue and auction list as arguments. It is from this thread where most processing takes place.

**Client Thread** – This class does most of the processing for the server. Once spawned, it first requires the user to login (loop 1) and then (upon successfully entering a valid username) it checks if any pending messages are waiting to be sent to that user. After this is completed, the process receives/processes input data from the client (loop 2). The input data consists of several commands depending on which client is connected. All responses are sent back to the client as well as informative messages that are sent back to the server. Some messages also send messages to other users (informing them of relevant updates). If that user is not online, the message is saved in the message queue until they next login. Once the /quit command is received, the second loop is terminated, the thread is closed and it is removed from the thread array. The final response from the server will also tell the client to shut down as well.

**Buyer/Seller** – Both programs function almost identically as described in the high-level explanation.

**Auction List** – Auctions are stored in a linked list. Each node contains the item number, item name, highest bid and highest bidder. The nodes are of type AuctionItem (see AuctionItem.java).

**Message Queue** – Undelivered messages are also stored in a linked list. Each node contains the recipient’s name and message. The nodes are of type Message (see Message.java).