CE303 – Stock Market Assignment

Daniel Woolsey

Implemented Functionality

Function	**C#**	**Java**
Client establishes a connection with the server	YES	YES
Client is assigned a unique ID when joining the market	YES	YES
Client displays up-to-date information about the market state	YES	YES
Client allows passing the stock to another player	YES	YES
Server manages multiple client connections	YES	YES
Server accepts new connections while traders are exchanging stock among themselves	YES	YES
Server correctly handles clients leaving the market	YES	YES
Client is compatible with the server in the other language	YES	YES
Additional tasks:		
Client GUI	NO	NO
Server GUI	NO	NO
Server restarts are correctly implemented	NO	NO
Unit tests	NO	NO

Protocol

Client Request	**Server Response**
(on connection) • Client is instantiated, waits for the server to return SUCESS message and their assigned id	SUCCESS <trader_id> • If they're the first client to connect, give them the stock
REFRESH • Return a String collection of the current state of the stock market • Even if this isn't requested, trades will work with any buyer as soon as they enter the market	<number_of_traders_in_market> foreach trader: <trader_id> : <stock_owned_boolean> …. ….
TRADE <seller_id> <buyer_id> • The client checks with the server that the seller owns the stock before allowing a trade request to be sent	SUCCESS FAIL: Buyer isn't in the market • Checked server-side, if buyer doesn't exist the trade will not be executed
CHECK • Get the id of the trader with the stock	<trader_id>

LEAVE	(handles disconnect) • Removes leaving trader from the stock market • If they own the stock, pass it to another market client

Client Threads

The main thread is the only thread that runs in the Client Process. Any requests made are validated by the server and one stock is tradable at any time, there didn't seem to be a need to spin a thread for every REFRESH or TRADE request.

Server Threads

The main thread (in TradingServer.java and Program.cs) will spin up a thread for every Client connection it receives. Each Client thread will only be terminated if they terminate their Client with an escape character or by choosing the menu option to leave. In both of these instances the Server will execute the finally block, where they are removed from the Stock Market data structure and their stock is given to another Client.

Project Review

I found the project to have a number of smaller challenges as more functionality was implemented. Handling Client disconnection meant blocking off any other Client actions until a number of situations were assessed: the toughest being passing the stock to another Client, achieved here by iterating through the trader map and checking for not null members. Another challenge was deciding where exceptions would be handled. Dealing with them in the Client meant only valid actions would be processed by the server, but some issues such as checking for a valid buyer id could be checked faster and more reliably by the Server.

I chose to use synchronized blocks to handle multi-threaded operations on the same data structures. This coarse-grained method of handling threads is many times slower than using a pre-existing thread-safe data structure (in this case, I would have used AtomicBoolean to manage the stock_owned variable. In this assignment we were asked to focus on reliability and usability, so I decided to sacrifice speed for simplicity and guaranteed safe from deadlock operation.

Managing this project between other assignments, modules and the 3rd year project has been the biggest issue for me. I put in abut an hour every few days for each stage of the assignment: writing up a protocol outline, re-studying the Bank example and writing the code. As for the additional tasks I chose to focus on the basic functionality rather than rabbit-hole into C# GUIs or serialising server information with GSON – both tasks that would require external libraries.