Document: Report

Creator: Dan Norstrom, dn18657, 1807572, NORST42202

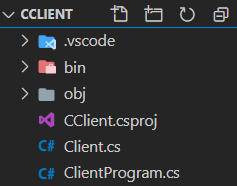
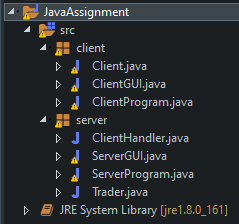
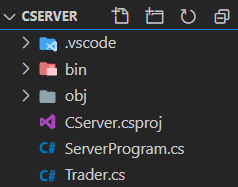
Course: CE303 Advanced Programming

Related to: Assignment

Date: 2020-11-28

|  |  |  |
| --- | --- | --- |
| **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** | **Yes** |
| Server GUI | **No** | **Yes** |
| Server restarts are correctly implemented | **No** | **Yes** |
| Unit tests | **No** | **No** |

**IMPLEMENTED FUNCTIONALITY**

**Protocol**

**Java & C# Application protocol:** Communication is carried out using sockets trough their respective readers and writers.

Every instance of ClientProgram (client-side) creates a socket-pair with a ServerProgram (server-side). Hence every pair where communication is carried out consists of a client-thread and a server-thread.

Once engaged, ClientProgram creates a new client and connect it to the server. Then it proceeds to call the .NewCient() method and run welcoming messages (C#) or start the GUI (Java).

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| .NewClient() | **NEWCLIENT** | **String customerID** |
| Description:   1. Client sends “NEWCLIENT” to Server 2. Server generates a unique ID as customerID and adds corresponding Trader to market. 3. Server returns String customerID to the client 4. Client receives customerID and adds it as a variable.   Purpose: Initialize client, add Trader to market, run welcome messages, make client aware of its ID    Picture[1]Java GUI Example of a new client | | |

**Java/C# Application protocol breakdown:** Once initialized successfully, the ClientProgram awaits User/GUI input. These are the following commands and their actions:

|  |
| --- |
| *Take note that CMD-input for the Java-version is deprecated and replaced with a GUI on both server and client side, so most of these messages are updated automatically by the ClientHandler.* |

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| .HasStock() | **HASSTOCK** | **Boolean** |
| Description:   1. Client sends “HASSTOCK” to Server 2. Server checks if the Clients ClientHandler has stock 3. Server returns a Boolean representing the clients stock status. 4. Client receives Boolean and reports/updates status   Purpose: Check if current trader has stock | | |

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| .GetStockOwner() | **STOCKOWNER** | **Int, String[ ]** |
| Description:   1. Client sends “STOCKOWNER” to Server 2. Server checks Market for Traders with stock 3. Server returns an int corresponding to the number of traders with stock 4. Client receives and saves this int. 5. Server returns int amount of strings corresponding to Traders with stock 6. Client receives int amount of strings and:  * Java: Color-codes stock owner in the GUI * C#: Displays stock owner in CMD/IDE-console   Purpose: Make client aware of who the owns stock. | | |

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| . GetTraders() | **TRADERS** | **Int, String[ ]** |
| Description:   1. Client sends “TRADERS” to Server 2. Server returns an int for the count of traders in the market 3. Client receives and saves this int. 4. Server returns int amount of strings corresponding to Traders in market 5. Client receives int amount of strings and:  * Java: updates GUI’s Active Trader List appropriately * C#: Displays Active Trader list in CMD/IDE-console   Purpose: Make client aware of all other traders in the market. | | |

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| .TradeStock() | **TRADESTOCK id1 id2** | **String** |
| Description:   1. If the client is a stockowner, the Client sends “TRADESTOCK id1 id2” where id1 is the receiver and id2 is the sender. Otherwise it reports to the user that its not a stock owner. 2. Server attempts to trade the stock. 3. Server returns appropriate messages such as “Success!” Or “Failure!” 4. Client reads and sends messages/ updates GUI appropriately   Purpose: Engage a transaction of stocks between clients. | | |

|  |  |  |
| --- | --- | --- |
| Client Method | Client Writer-> Server writer | Server Response |
| .ReconnectClient(  TraderID, cgui.hasstock) | **RECONNECTCLIENT id Boolean** | **String** |
| Description:   1. Client sends “RECONNECTCLIENT TraderID StockStatus” to the server after its restarted. 2. Server rebuilds Market-state based on the connected clients.   Purpose: Reconnects client to a restarted server with its original state. | | |
| Picture[2]: Java GUI Restart | | |

**Further Notes on Java-GUI:** The JList component colouring displays all the required information asked for in this assignment:

|  |  |
| --- | --- |
| Green | The current Client has stock. |
| Blue | The current Client. |
| Red | Client with Stock. |
| **Picture [3] : Java Server with 4 Java Clients** | |

**Java Client Threads**

**ClientHandler Main-Thread**

**Starts:** When Client process is started

**Ends:** When client is shut down

**Purpose:** Starts and initializes Client. Initializes and updates GUI with a While loop.

**Swing EDT (Event Dispatch Thread): Automatic**

**Starts:** Whenever we create a frame object using swing.

**Ends:** When the relevant swing frames are disposed.

**Purpose:** EDT runs ActionListener’s, Repaints and other events triggered by the user.

**C# Client Threads**

**ClientProgram Main-Thread**

**Starts:** When Client process is started

**Ends:** When client is shut down

**Purpose:** Starts and initializes Client. Reads user inputs and writes server responses in CMD / IDE-console.

**Java Server Threads**

**ServerProgram: Main-Thread**

**Starts:** When server process is started.

**Ends:** When server is shut down.

**Purpose:** Starts and initializes server. While-loop listens for new client connection and creates ClientHandler-threads for them.

**ClientHandler-Thread**

**Starts:** When a new client connects.

**Ends:** When the client disconnects.

**Purpose:** Handles communication between the server and client. Listens and writes to/from its client.

**ServerGui Anonymous Update-Thread**

**Starts:** When ServerGui initializes.

**Ends:** When server is closed.

**Purpose:** Updates Server GUI’s fields.

**Swing EDT (Event Dispatch Thread): Automatic**

**Starts:** When ServerGui initializes.

**Ends:** When server is closed.

**Purpose:** EDT runs ActionListener’s, Repaints and other events triggered by the user.

**C# Server Threads**

**ServerProgram: Main-Thread**

**Starts:** When server process is started.

**Ends:** When server is shut down.

**Purpose:** Starts and initializes server. While-loop listens for new client connection and creates HandleIncomingConnection-threads for them.

**ClientHandler-Thread**

**Starts:** When a new client connects.

**Ends:** When the client disconnects.

**Purpose:** Handles communication between the server and client. Listens and writes to/from its client.

**Project review**

The Project went well, handling concurrency was particularly challenging, while translating java-code to C#-code was surprisingly simple.

I’m particularly proud over the compact Swing JList-implementation and the swift restart functionality.

There might still be hidden concurrency-issues, it was extremely engaging to test this “Black-box”-like socket-application. Some things that happened but haven’t occurred again after fixes include:

* String Output intertwined.
* JList-component broke on 30+clients. Repaired automatically on server-restart. Suspecting java.util.NoSuchElementException on list-element.
* ConcurrentModificationException in synchronizedLIst().

Project management was stressful and time-consuming with another deadline and capstone in parallel.

If I redid this assignment, I would have used UML to plan threads/communication ahead of time, saving myself a lot of work.