E/CS 4390.003/5390.003 – Computer Networks
Fall 2022
Team Project: Team 24
Due: December 06, 2022 @ 11:59PM

Simple Math Calculation Client and Server Chatbox

Group Members:

- Keegan Kimbrell (kmk170830)
- Junjie Lin (jxl180101)
- Aroofa Mohammad (aam170007)

Protocol Design:

Our protocol for the message format is as follows:

1. Client sends an initial request for connection to the server along with a message that represents the name of the client.
2. After the connection, the server sends two messages. It sends
   a. ”A new user has connected to the server!” to the output.
   b. ”SERVER: ClientName has joined the server!” To all other connected clients.
3. After the initial connection all messages sent from the client to the server are then relayed by the server to each client. For some message sent to server, the server relays:
   a. “ClientName: message” to all other clients.
4. If this message represents a math equation, the server will respond to the original client with the result of the equation in the form:
   a. “SERVER: Equation Answer”.
5. Upon receiving a request ‘quit’ to close the connection, the server will send a message to all other clients in the form of:
   a. “Server: ClientName has left the server!”

The format for receiving a math equation is as follows:
When a message is received, if it is in the form of (spaces in between the components):

1. NUMBER OPERATOR NUMBER
2. Then the server will determine this as a math equation and process the request. The answer will be returned in the form of a double.

For example the request:
“5 * 15”
Will return the response:
“SERVER: 75.0”

The protocol for logging the information of the user is as follows:

1. Upon client connection to the server, the server will log:
    a. "ClientName StartTime: WeekDay Month Day Time Timezone Year"
2. Upon client disconnect with the server, the server will log:
    a. "ClientName EndTime: WeekDay Month Day Time Timezone Year"
    b. "ClientName Time Spent: Hours: # Minutes: # Seconds: #"

Programming Environment Used:

A multitude of programming environments were used to ensure the robustness and accuracy of the program. The program was created and tested over using Java within Eclipse, Microsoft Visual, and the command line. The program was further tested on Linux machines.

How to Compile/Execution Program:

(Preferred Execute Method):

Linux:

1. If makefile.xxx, then in comment line type "mv makefile.xxx makefile" to rename the file
2. In command line type "make" to compile all the java files
3. In command line type "java Server" to run Server
4. In command line type "java Client"

Windows Command Line:

1. -javac -cp . *.java
2. "java Server" to run Server
3. "java Client" to run Client

Parameter needed during execution(i.e IP, port, maybe name):

No parameters needed for execution, the program runs on port.1234 on the localhost.

If not working what:

Our code fulfills all aspects of the requested functionality.

Challenges Faced:

When running the program in eclipse some issues were faced with the accept() method, and the client would fail to connect to the server. Another problem in eclipse was when you ran the server on a specific port number that port number would still be established when trying to rerun the program resulting in an error where the programmer has to terminate the process or restart their personal computer because the code will not run on a

used port. After researching the issue, the team learned eclipse is not the best environment for client and server applications. Rather preferred environments were visual studio, linux, and cmd.
When creating the ClientHandler the team encountered issues with the program crashing. After debugging using print statements, we learned that socket ‘s’, BufferReader, and BufferWriter were not closing at the right time. Which cause error threading error and infinity loop, which were fixed by correctly closing sockets and I/O stream in and around the while loop.

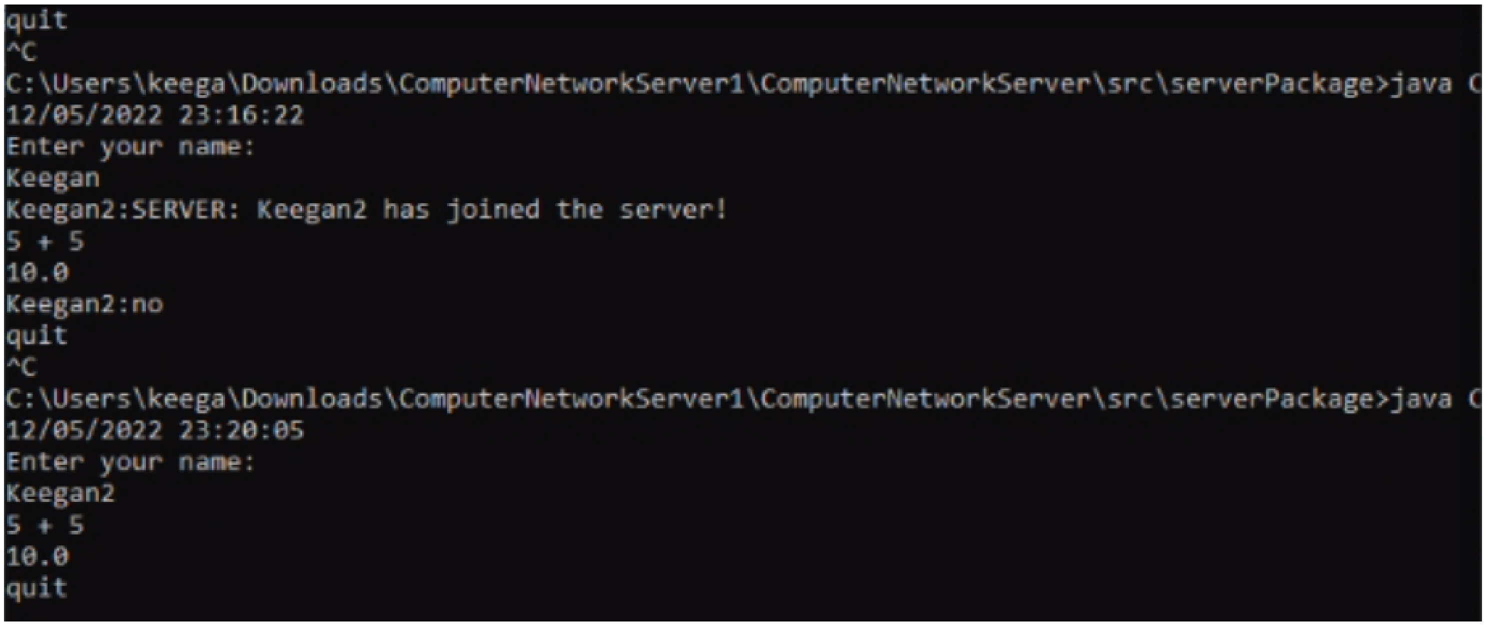
Lesson Learned:

This computer network project taught the team how to implement networked applications using sockets. A socket connection is required in order to connect to another machine. When two machines are connected(via a socket), they are aware of each other's IP address and TCP port on the network. Streams are used for data input and output to communicate information through a socket connection. Two sockets are required in Java to write a server application, ServerSocket (wait for client request) and a normal Socket (for client communication). In order to connect with a server, a client program creates a socket at its end. On the communication end, the server creates a socket class object once the connection has been made. Now that the socket has been established, the client and the server can exchange data.
The first step in creating a TCP connection between two devices is the server defining a ServerSocket object and specifying the port number (where communication will take place). The server requests the accept() function of the ServerSocket class after creating the ServerSocket object. Until a client connects to the server on the specified port, this application pauses. A client creates a Socket class object after the server has been idle, specifying the server name and port number to connect to. Following the aforementioned action, the constructor of the Socket class tries to connect the client to the specified server and port. In the event that communication is authenticated, the client gets control of a Socket object for communicating with the server. In the server the accept() function will return a reference to a new socket which is connected to the socket of the client.
I/O streams can be used for communication once the connections have been established. An output stream and an input stream are both present in every object of the socket type. The server's InputStream and the client's OutputStream are combined, and the client's OutputStream is connected with the server's InputStream. A two-way communication protocol is the Transmission Control Protocol (TCP). As a result, data can be transferred via both streams at the same time.

Output Screenshots:

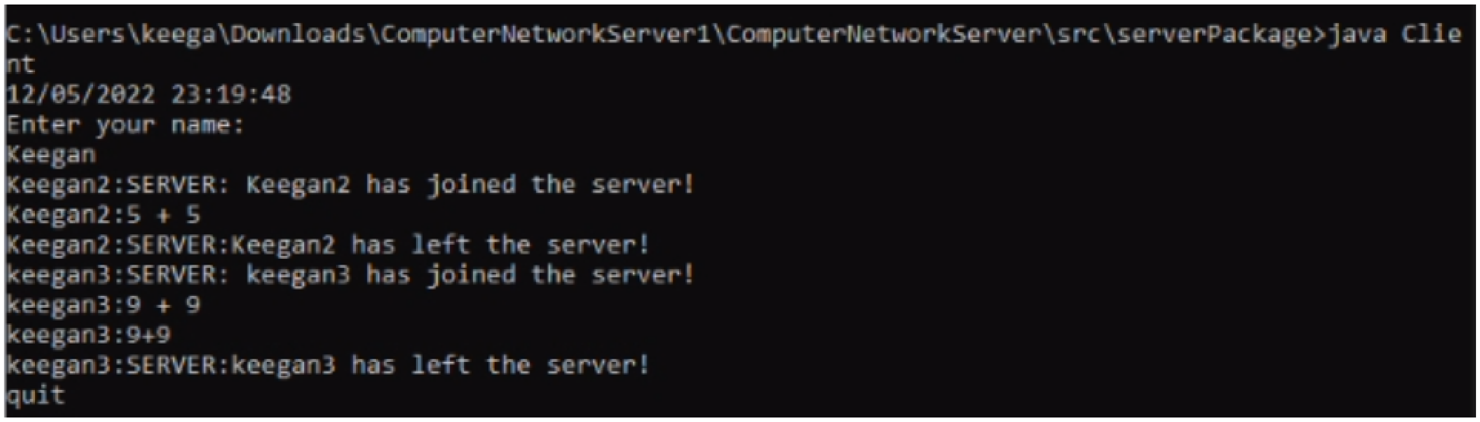
The following two screenshots show that multiple clients can join the server:

First Client:



```
quit
^C
C:\Users\keega\Downloads\ComputerNetworkServer1\ComputerNetworkServer\src\serverPackage>java C
12/05/2022 23:16:22
Enter your name:
Keegan
Keegan2:SERVER: Keegan2 has joined the server!
5 + 5
10.0
Keegan2:no
quit
^C
C:\Users\keega\Downloads\ComputerNetworkServer1\ComputerNetworkServer\src\serverPackage>java C
12/05/2022 23:20:05
Enter your name:
Keegan2
5 + 5
10.0
quit
```

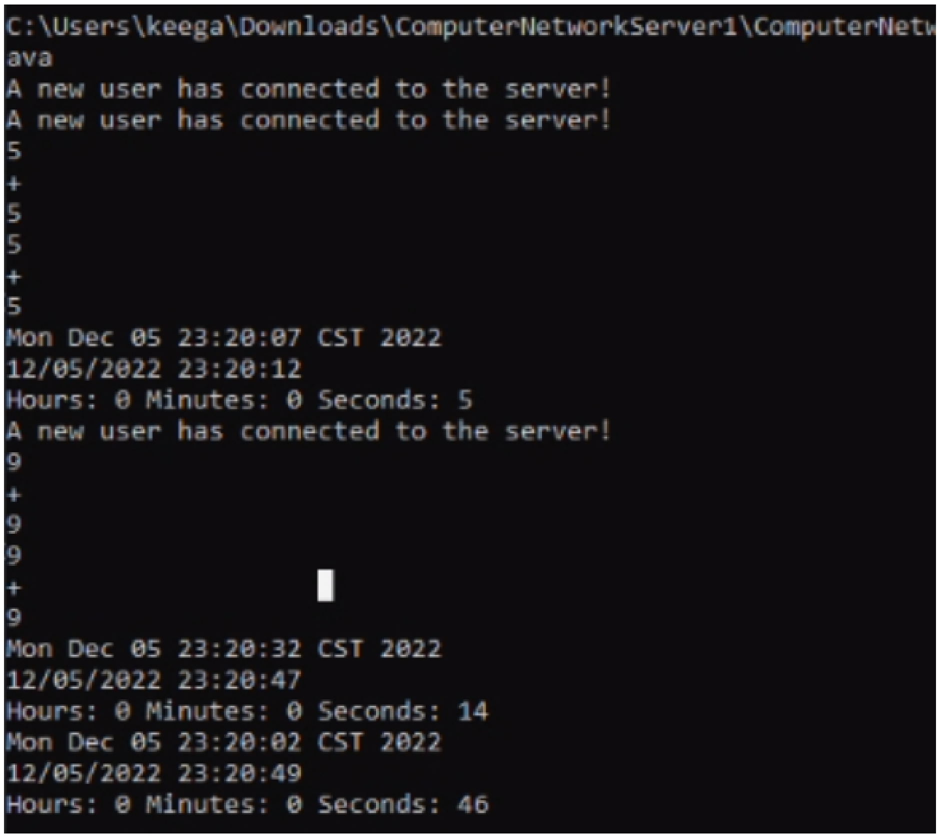
Second Client:



```
C:\Users\keega\Downloads\ComputerNetworkServer1\ComputerNetworkServer\src\serverPackage>java Clie
nt
12/05/2022 23:19:48
Enter your name:
Keegan
Keegan2:SERVER: Keegan2 has joined the server!
Keegan2:5 + 5
Keegan2:SERVER:Keegan2 has left the server!
keegan3:SERVER: keegan3 has joined the server!
keegan3:9 + 9
keegan3:9+9
keegan3:SERVER:keegan3 has left the server!
quit
```

The following screenshot shows the server being able to connect to multiple clients:

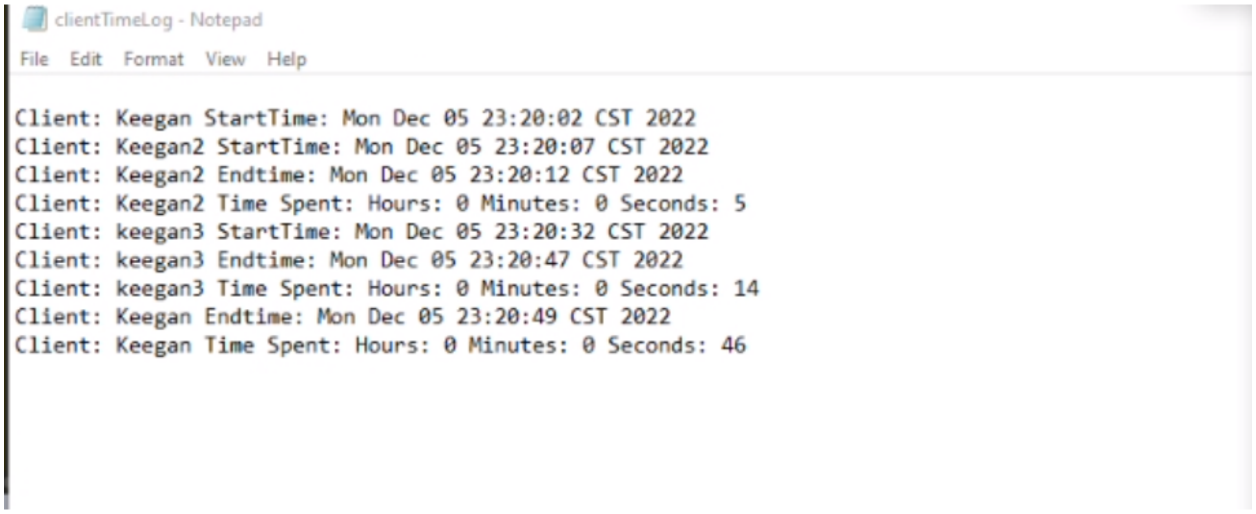
Server:



```
C:\Users\keega\Downloads\ComputerNetworkServer1\ComputerNetw
ava
A new user has connected to the server!
A new user has connected to the server!
5
+
5
5
+
5
Mon Dec 05 23:20:07 CST 2022
12/05/2022 23:20:12
Hours: 0 Minutes: 0 Seconds: 5
A new user has connected to the server!
9
+
9
9
+
9
Mon Dec 05 23:20:32 CST 2022
12/05/2022 23:20:47
Hours: 0 Minutes: 0 Seconds: 14
Mon Dec 05 23:20:02 CST 2022
12/05/2022 23:20:49
Hours: 0 Minutes: 0 Seconds: 46
```

The following screenshot shows the server log events of when a client were connected, disconnected, and the total time they spent on the server:

Log:



```
Client: Keegan StartTime: Mon Dec 05 23:20:02 CST 2022
Client: Keegan2 StartTime: Mon Dec 05 23:20:07 CST 2022
Client: Keegan2 Endtime: Mon Dec 05 23:20:12 CST 2022
Client: Keegan2 Time Spent: Hours: 0 Minutes: 0 Seconds: 5
Client: keegan3 StartTime: Mon Dec 05 23:20:32 CST 2022
Client: keegan3 Endtime: Mon Dec 05 23:20:47 CST 2022
Client: keegan3 Time Spent: Hours: 0 Minutes: 0 Seconds: 14
Client: Keegan Endtime: Mon Dec 05 23:20:49 CST 2022
Client: Keegan Time Spent: Hours: 0 Minutes: 0 Seconds: 46
```

Design Document

Main parts of the program:

1. Server Socket(Server.java): The Server Socket is responsible for processing requests over the network, it serves as one end of the two way communication over the network. For instance, the server socket takes care of the connection request from multiple clients.
2. Clients(Client.java): The client is responsible for sending the connection request and termination request, it also serves as the other end of the two way communication. The input of the client is stored in the buffer writer while the output of the bufferedwriter is being broadcast to all the clients that are being connected on the same server. This accomplished the group chat feature, and if the ClientHandler deemed that the output from the bufferedwriter is a math equation, this is where the math server kicks in, which the server will return the answer of the math problem and the clients can see this by listening to the bufferedreader.
3. Client Handler(ClientHandler.java): The client handler serves as the middle man in the two way communication. It takes care of the client input and decides whether it’s a group chat message that needs to be broadcast or a math problem that needs to be solved. The client handler is structured in a similar fashion as the client as it’s mainly dependent on a bufferedreader and bufferedwriter to interact with other parts.