**CSC3002F: Assignment 2A**

**7 April 2017**

**Group Members:**

**Imaan Dolley (DLLIMA001)**

**Breyden Monyemoratho (MNYBRE002)**

**Kushal Cuppoor (CPPKUS001)**

**CSC3002F: Assignment 2A**

**Group: Imaan Dolley (DLLIMA001), Breyden Monyemoratho (MNYBRE002), Kushal Cuppoor (CPPKUS001)**

**Server/Client Chat Overview:**

The **Server** class receives messages and files from clients and sends them to the other clients connected to the server. The **Client** class represents someone connected to the server who is able to chat to other connected clients.

To ensure that each client can send and receive messages simultaneously, several threads are started, namely **MessageReaderForServer**, **MessageForwarder**, **MessageSenderToServerForClient**, **FileForwarder**, **MessageReaderFromServerForClient**

The **MessageReaderForServer** class reads and prints to the server any messages that has been received from accepted clients. The **MessageSenderToServerForClient** class reads output from the client and sends the message to the server. The **MessageReaderFromServerForClient** class is used to receive messages from the server and print them out for the client. The **MessageForwarder** class is used by the server to send a message to all connected clients apart from the client who sent it. **FileForwarder** is a class used in the same manner as the **MessageForwarder**, however, instead of only Strings being sent, files or images can be sent to clients when “@sendfile:” is typed before the file’s path.

**How the Classes Operate:**

The server is started first and the number of people joining the chat as well as their individual port numbers are required as input. Once the port numbers are entered, the port numbers and server sockets are each placed into arrays. Then the server starts and the clients can each request a connection to the server. A server socket is created for each client that connects to the server. The server uses server sockets to accept connection requests from clients. The **Server** class starts threads of the **MessageReaderForServer** class for each client so that the Server can simultaneously read messages and accept files from all clients.

The **MessageReaderForServer** class checks if the sockets are closed before doing anything in case the thread has been interrupted. If the socket is still open, it notifies the server that a client wants to send a message or file. The message is read using an InputReader and a **MessageForwarder** thread is started. If the message contains “@receiveFileRequest:” with a path, the server knows the client is trying to send a file and receives the file. If there is an error with the path, a default path is set. The file is queued for transfer to other clients but the server waits until the other clients accept the file by typing “@acceptfile”. Images/files a stored on the server if the recipient accepts it. However, if the clients reject the file, the server does not store the image/file. If the clients accept the file, a **FileForwarder** thread is started to send the file/image. If the clients reject the file by typing “@rejectfile”, the server is notified and the file is not sent to those clients. If two clients want to have a private chat, they can type the “@private” followed by the receiver’s port number and then the actual message.

**FileForwarder** is a class which notifies the server that a client has requested transfer of a file. The class is called in the **MessageReaderForServer** class when clients accept a request for a file transfer. The file is sent as bytes from the server to those clients. The client is notified when the image has been sent.

When a **Client** class is opened, the port number must be entered so that the server can assign a port number to a particular client and so that during the chat, the server knows which client has sent a message or file. The server and all clients currently connected are notified when a client joins the chat. While the client is connected to the server, threads for the **MessageReaderFromServerForClient** class and **MessageSenderToServerForClient** class are started. These classes allow the client to read messages received from the server as well as read messages which are sent to the server.

The **MessageReaderFromServerForClient** class is used by the **Client** class to read messages that the server has received and sent from other clients. This class reads the outputstream from the server and the message is printed out to the client. If the message is “@receiveFileRequest”, the file is created in the received\_files directory for that client by passing the file as bytes

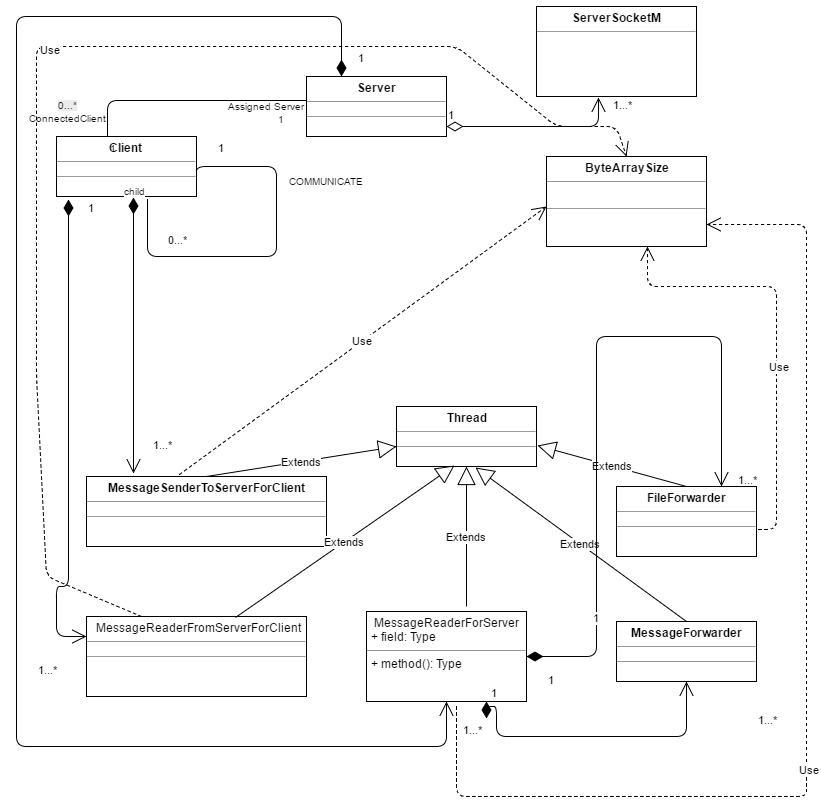
The **MessageSenderToServerForClient** class is used by the **Client** class to read messages that the client has sent and send it to the server using an outputstream. If the message is @exit, the client can exit the chat. If the message contains “@sendfile” with a specified path, the client wants to send a file to the other clients via the server. The server is notified if a client wants to send a file and the file is then sent as bytes to the server using a byte array.

The class **MessageForwarder** is used by the **Server** class to send messages from one client to all other clients. Several threads of this class exist for every message sent by connected clients. This ensures that messages from all clients can be sent simultaneously. The class runs a loop through all the connected clients and send the message to all clients except the sender.

**ByteArraySize** is used to give the size of the byte array in kilobytes when a file is transferred in a chat. Using a class makes it easier to change the byte array size which is used in more than one class.

When a client leaves the chat by typing “@exit” or the program is ended, the server and the other clients are notified by message, all of the threads pertaining to the sending and receiving of messages for that client are interrupted and the input/output streams, sockets and server socket for that client are closed. No messages can be sent or received by this client anymore as the connection to the server is terminated. If the server disconnects, the same events occur and the clients are notified that the server has disconnected.

**Class Diagram:**



**Screenshots:**

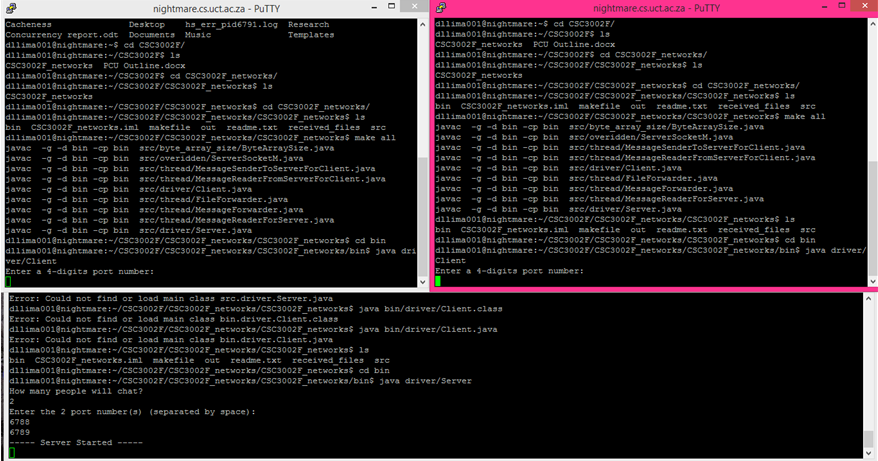
Figure 1: Start of the Server Client Chat Application

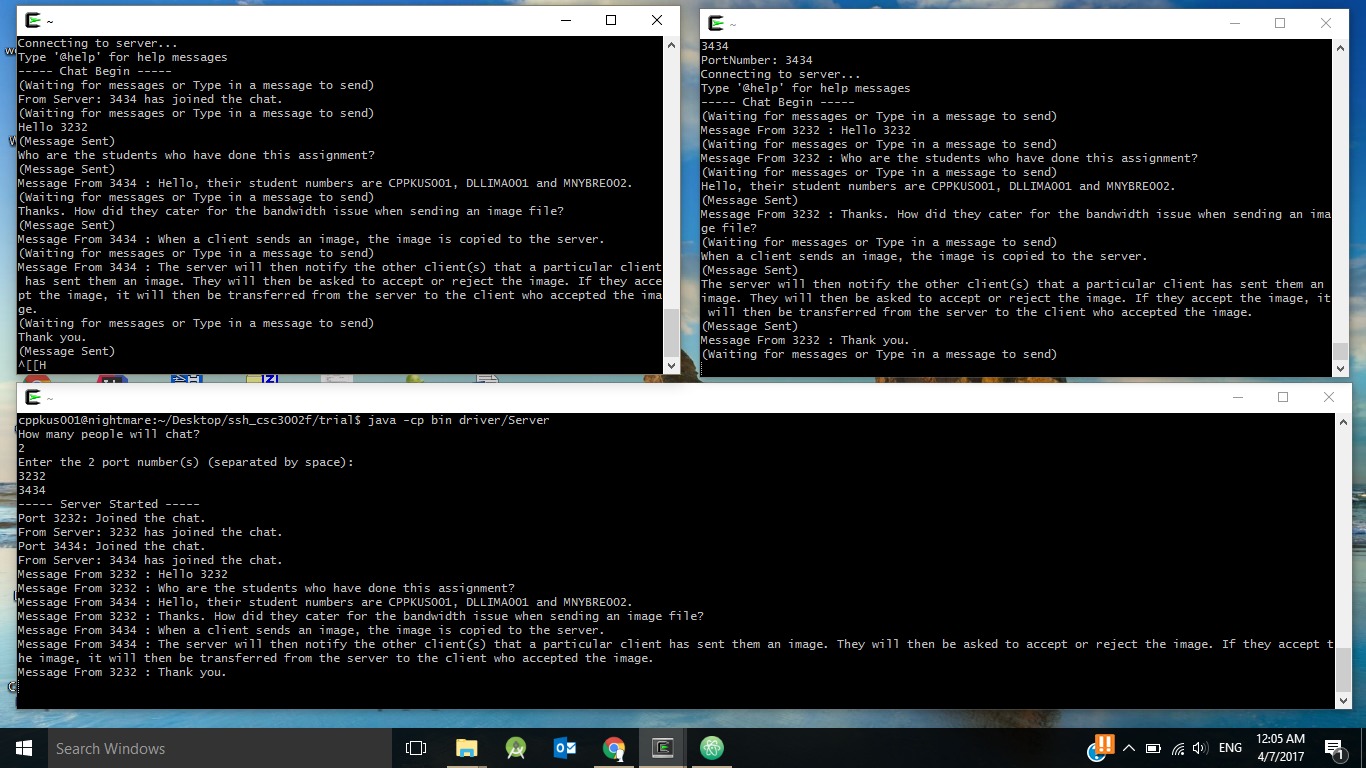
Figure 2: Example of the Server Client Chat Application

Figure 3: Demonstration of How File Sharing Process Works