**Lab 4 – Networking**

**Chat Room**

**Lab Objectives**

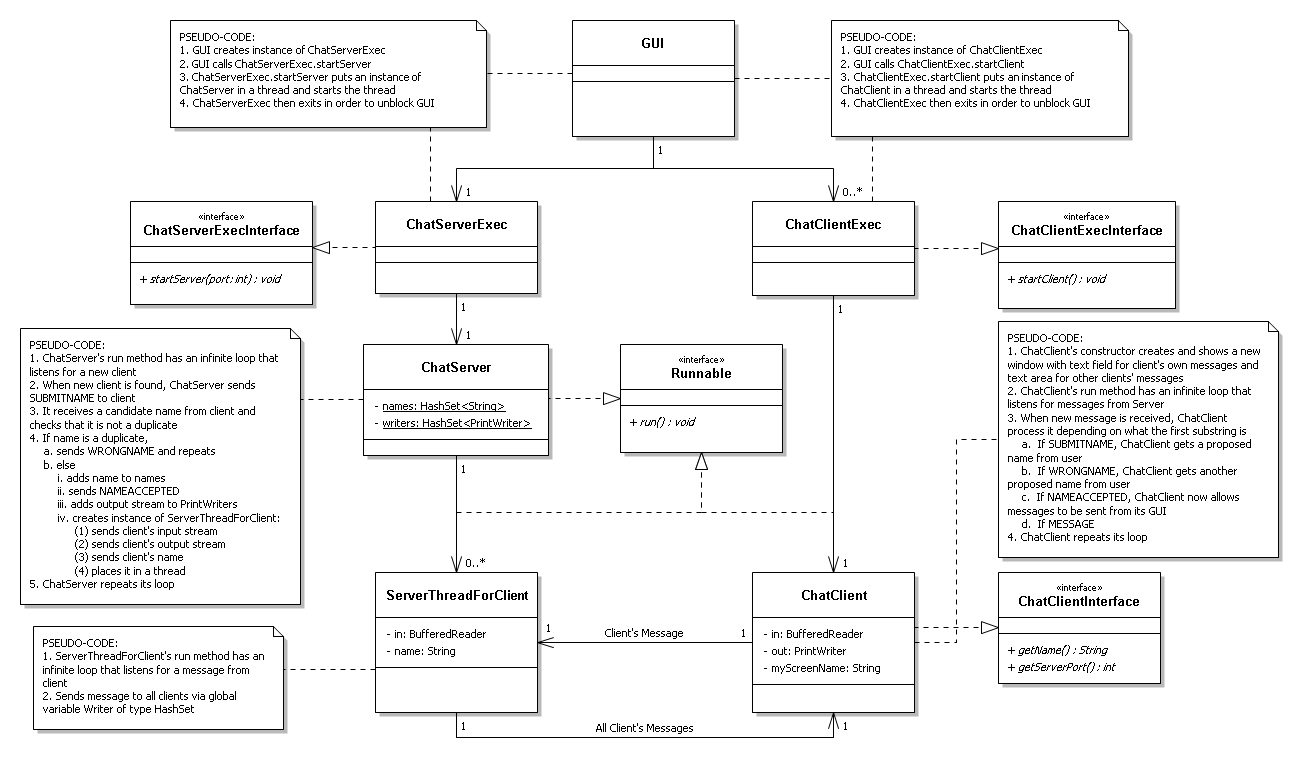
* Be able to create server and client sockets
* Be able to create input and output streams to and from sockets
* Be able to create threads
* Be able to run an object in a thread

**Introduction**

Everyone is familiar with chat rooms. In this lab we are going to create a server/client infrastructure that supports multiple users sending messages to each other.

In this lab you will not need to build the user interface, or to design the class structure. You will be given most of the class structure and will focus on creating instances of classes that run inside threads, and on creating sockets and in- and out-streams with which to communicate.

First, we need a blueprint. Here is the basic class diagram:

****

Let’s look more closely at each class.

1. *GUI*. The user interface is a set of classes providing the basic interface to start a server, start each client, and exit the application. You will not need to edit this code.
2. *ChatServerExec*. The sole responsibility of the *ChatServerExec* is to implement the method *startServer*. This class implements the *ChatServerExecInterface*. You will be editing this class to put the server in a thread and start the thread. The ChatServer is placed in its own thread so the GUI can operate asynchronously. The ChatServerExec then exits, while the ChatServer continues to operate.
3. *ChatServer*. The responsibility of this class is to create the server socket, listen for each client, and set up in- and out-streams. It then ensures that each client submits a unique screen name and adds the name to the set of client names. Finally, it creates a server to respond to the individual client and places it in a thread. The *ChatServer* than continues its loop to listen for the next client.
4. *ServerThreadForClient*. This server responds to an individual client. When its client sends it a message, this server is responsible for sending out that message to each out-stream, i.e., to each client. Note that this is an inner class.
5. *ChatClientExec*. Just like the *ChatServerExec*, this class is responsible for implementing *startClient*, which involves running the *ChatClient* instance in a thread. This class implements the *ChatClientExecInterface*.
6. *ChatClient*. This class is responsible for building its own window (code provided) as a separate GUI. You will create a client socket with server address and port and set up in- and out-streams to the server. The class then follows the application protocol. This involves first getting a screen name from the user, and then receiving messages from the clients via the *ServerThreadForClient* instances and posting the message in its textarea. When this client sends a message, the user enters it into the window’s textfield whereupon it is sent to its *ServerThreadForClient. ChatClient* implements *ChatClientInterface.*
7. Upload the initial files in GitHub in the repository you created in Lab 1, and modify them as you implement the solution. Take a screen shot of the repository and submit it.

**Task #1 Starting a Thread**

1. In *ChatServerExec*, create a thread containing the server instance, and start the thread.==> done
2. In *ChatServer*, create a server socket. Also uncomment the System.out.println so that the console reflects that a server was started.
3. When you run the GUI, you will be able to start the server, and launch clients, but will not be able to enter anything in the client’s textfield.

**Task #2 Creating In- and Out-Streams**

1. Inside the *ChatServer* while loop, listen for a client to join, and when one does, create an in- and an out-stream.
2. Compile and run, but you will still not be able to enter messages.

**Task #3 Creating the Client**

1. In *ChatClientExec*, create a thread containing the client instance, and start the thread.
2. In *ChatClient*’s run method, create a client socket with server address and server port.
3. In *ChatClient*’s run method, set up in- and out-streams.

**Task #4 Running the application**

1. Compile and run.
2. Start the server first.
3. Then start a client. Enter a screen name.
4. Start a second client, with a unique screen name.
5. Send messages by entering in a client’s textfield.
6. Take a screen shot of two clients running and chatting.