Distributed Systems – Internet Chat System
Project documentation

Project documentation content:

Software Engineering Requirement:

An internet chat system has two basic components: A server and a client.

We will analyze the requirement as is applicable to the system as a whole as well as for each component. We are provided with the TCP protocol to connect our system.

Functional Requirement for the Server:

- A server should run on a specified port.
- It should listen for and accept connections from clients over a TCP connection.
- It should maintain a list or collection of such clients.
- It should receive data from the client and pass it to all the other clients.

Functional Requirement for the Client:

- Client should try to connect to the server when given an ip address and a port number.
- It should receive messages from the server and display it.
- It should read data from the keyboard and send it to the server.
- It should not mix up the users own typing and the message received from the server.

Non Functional Requirement:

Language and library:

The Java 1.5 API and only standard Library will be used.

Efficiency for Server:

The Server must be able to handle simultaneously a large number of clients. A 300 clients maximum limit would be appropriate. It must be able to run on a modest hardware, like the following:

Processor: 1.6 GHz

Memory: 1 GB ram

Network bandwidth 100Mbps

Efficiency for Client:

Any java virtual machine capable of running a TCP protocol and the command line, eg. smartphones, laptops, thin clients.

Reliability

This section describes the reliability of the server and the client.

Reliability for Server:

The server must be able to manage its system resource. It must create a lot of connections and close them without running out of resources.

The connections may break, and the server must recover and reuse the resource. The server must not block on any of the connections.

Reliability for Client:

A problem with a single client must not affect the other users connected to the server.

Security:

The system opens a serversocket and large number of sockets on a network. Therefore it is imperative that the sockets are protected from outside malicious users.

It may implement an authentication and/or data encryption.

Priority:

Must have:

- Open a Server Socket on the server
- Add clients into a list on the server
- Open a socket on the client and on the server
- Send a message to the Server from the client
- Receive a message from the Client to the server
- Send copy of a message to all the Clients on the server
- Drop a client whose socket has been blocked or closed

Should:

- Server should only allow legitimate chat clients to establish connections.
- When the user is typing, incoming messages must be buffered in order to avoid any overlap. They will be displayed later, once the user has entered his/her message.
- Server should manage the nick names and prevent duplicates.

May:

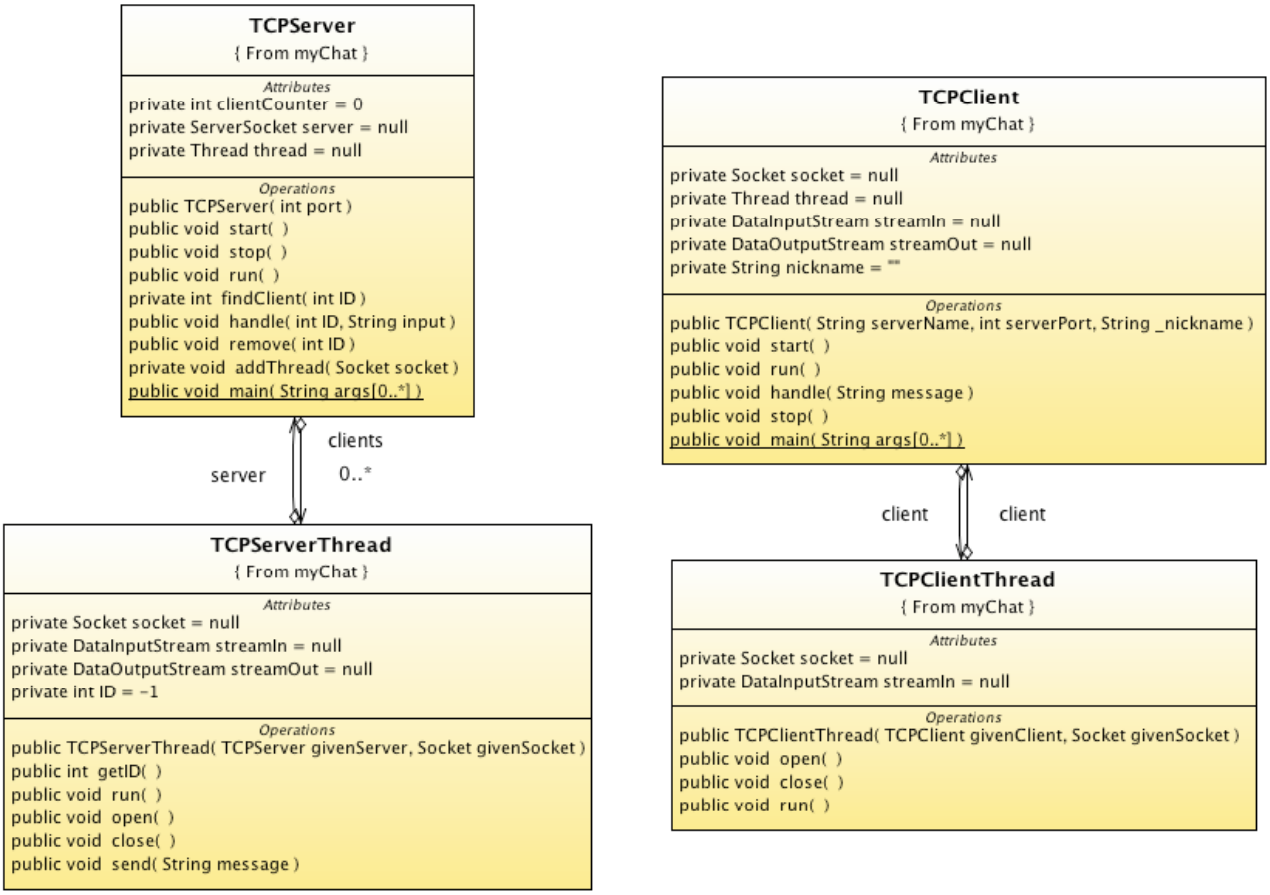
- Server and client should encrypt the data.
- The chat client may use a GUI.
- The chat client may implement a filter so that only selected users message are displayed.

Algorithm design

This section describes the sequence of the program, shows the UML diagram and the classes being used.

Description of sequence / flow chart:

UML diagram:



Classes:

There are the flowing classes being used from the server:

- TCPServer.java
- TCPServerThread.java

There are the flowing classes being used from the client:

- TCPClient.java
- TCPClientThread.java

Comparison With a UDP implementation:

Installation and Usage Guide:

For running the server as well as the client we created JAR files that can be used in the following way.

Usage of "server.jar":

- Use the command line and be sure that you have the "server.jar" file in the same directory where you like to run it, if you are not currently there.
- Then you can launch the server with the following command:
  - `java –jar server.jar`
- If you get the following output, the server is correctly running:

```
Server: doing the set up ...
Server: setting port at 7896
Server: started server on
ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=7896
]
Server: waiting for clients ...
```

- You can stop the server by typing ^C

Usage of "client.jar":

- Use the command line and be sure that you have the "client.jar" file in the same directory where you like to run it, if you are not currently there.
- Then you can launch the server with the following command:
  - `java –jar client.jar`
- If you get the following output, the client has correctly started:

```
Welcome to CommandLineChat
==========================

Enter server IP:
```

- As next you are asked to enter the IP of the server where the “server.jar” file is running (followed by enter).
- As next you are asked to enter the nickname you wish like to use (followed by enter).
- If you get the following output, the client has correctly connected to the server:

```
Trying to establish connection to server localhost on
port 7896.
Please wait ...
Perfect, connection to server localhost properly
established!

You can now start typing a message.
```

- Then you can go on using the command line chat and for leaving the chat you can type “/exit”.