Exercise 08: Threads, IO, and Networking

Due on:19. 05. 2016, 10:00

Name: ________________________________ **Studentnumber**: ____________________

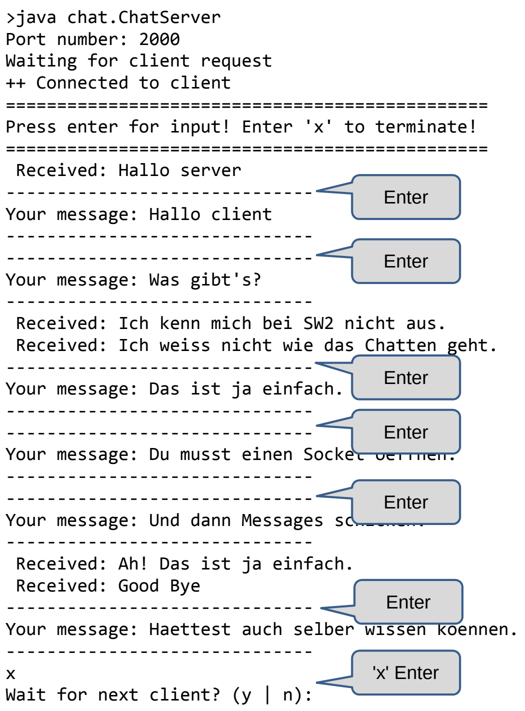
CS: ☐ G1 (Grimmer) ☐ G2 (Prähofer) ☐ G3 (Prähofer)

WIN: ☐ G1 (Khalil) ☐ G2 (Khalil) ☐ G3 (Hummel)

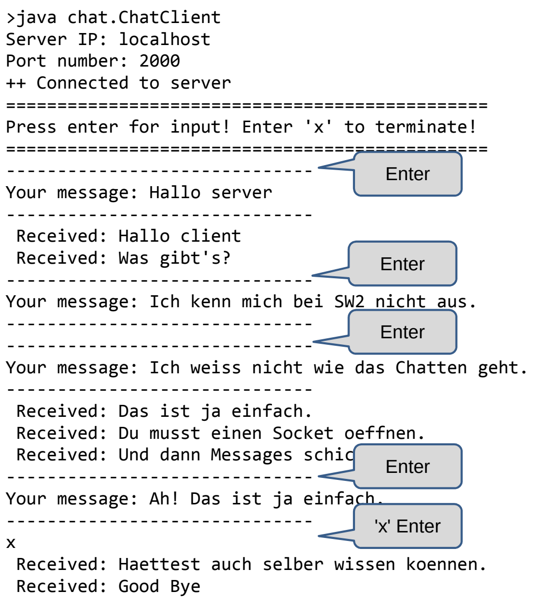
Task	Credits	Hand in			
Exercise 08	24	Java-Program, output of a test run, UML diagrams		☐	

ChatServer and ChatClient (20 Points)

Create a simple chat application that supports a chat between two participants over the Internet using threads, sockets, and streaming. A possible procedure is outlined in the following (messages in German).



```
>java chat.ChatServer
Port number: 2000
Waiting for client request
++ Connected to client
===============================================
Press enter for input! Enter 'x' to terminate!
===============================================
 Received: Hallo server
------------------------------
Your message: Hallo client
------------------------------
------------------------------
Your message: Was gibt's?
------------------------------
 Received: Ich kenn mich bei SW2 nicht aus.
 Received: Ich weiss nicht wie das Chatten geht.
------------------------------
Your message: Das ist ja einfach.
------------------------------
------------------------------
Your message: Du musst einen Socket oeffnen.
------------------------------
------------------------------
Your message: Und dann Messages schicken.
------------------------------
 Received: Ah! Das ist ja einfach.
 Received: Good Bye
------------------------------
Your message: Haettest auch selber wissen koennen.
------------------------------
x
Wait for next client? (y | n):
```

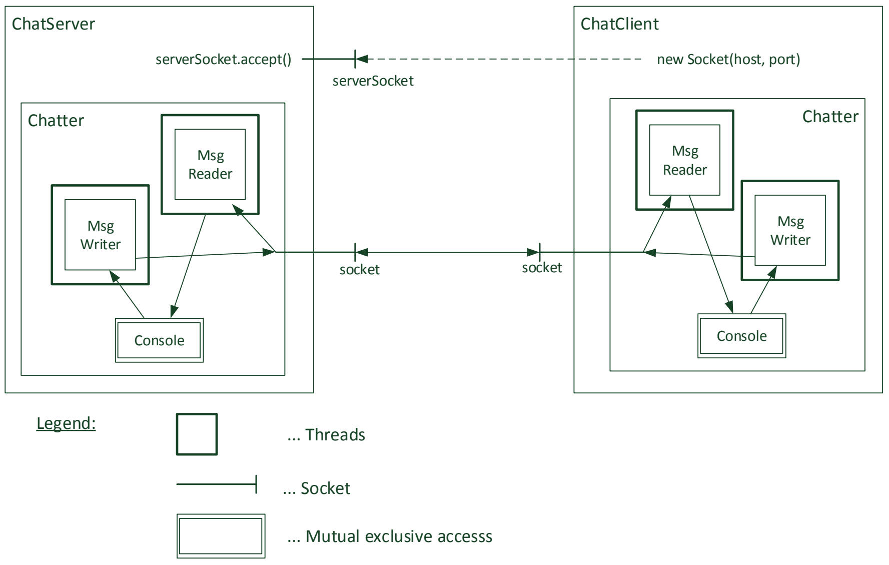


```
>java chat.ChatClient
Server IP: localhost
Port number: 2000
++ Connected to server
===============================================
Press enter for input! Enter 'x' to terminate!
===============================================
------------------------------
Your message: Hallo server
------------------------------
 Received: Hallo client
 Received: Was gibt's?
------------------------------
Your message: Ich kenn mich bei SW2 nicht aus.
------------------------------
------------------------------
Your message: Ich weiss nicht wie das Chatten geht.
------------------------------
 Received: Das ist ja einfach.
 Received: Du musst einen Socket oeffnen.
 Received: Und dann Messages schic
------------------------------
Your message: Ah! Das ist ja einfach.
------------------------------
x
 Received: Haettest auch selber wissen koennen.
 Received: Good Bye
```

- A user starts a ChatServer using a port that is entered on the command line (port should > be 2000).
- Another user starts the ChatClient, reads the server address (here, "localhost") and the port from command line and connects to the server.
- The chat participants press the Enter key to send a message. Then, a line can be entered that will be sent to the respective other participant.
- Participants receive the messages of the other participant and print them to the console.
- With the input of "x", a participant can terminate the chat. Chat termination should be processed as follows:
  - Send message "Good Bye". After sending, this participant cannot send any further messages.
  - The chat partner can still send more messages which are received and printed out.
  - The chat partner must enter "x" to close the chat also from his/her side. Then, both sites terminate.
- The ChatServer asks for user input with " Wait for next client? (y | n): ", in case of “y” (yes), the ChatServer should wait for a new connection request from a client.

Architecture

The following illustration shows the architecture of the system. The ChatServer waits for connections and the chat client initiates a connection. Then, the communication is handled via chatter objects on both sides. Reading the messages from the socket, and writing to the console happens in a thread, entering of the message, and sending to the socket happens in another thread. Access to the console is mutually exclusive (i.e., only one thread is allowed to access the console at a time).



Notes:

The following requirements have to be met:

- Receiving and sending of messages must be carried out in two different threads.
- It is important that the message output to the console and the input from the console happen mutually exclusively (otherwise, received messages would be mixed with entered text). Therefore, you must encapsulate the input and output to the console in its own object, and synchronize.

  Note: Pressing the Enter key and "x" should not be synchronized, because otherwise the console remains always locked.
- Make sure to finish the chat correctly at the client and the server. On entering 'x' the following should be implemented:
  - Send the completion message "Good bye"
  - Stop the chat with socket.shutdownOutput();
  - The thread reading new messages continues until also the other side does not send any messages any more
  - The other threads waits for the reading thread before it terminates (hint: join())
  - Finally, close the socket and terminate.

Diagrams (4 P)

Draw the following two diagrams:

- Class diagram: Draw an UML class diagram
- Sequence diagram: Draw the process of message sending when A sends a message, B receives the message and answers, A ends the chat, B ends the chat.

  Manual drawing is sufficient.