technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 4

Dr. Jan Kriege
Dipl.-Inf. Horst Schirmeier

Winterterm 2015/16

Computer Networks and Distributed Systems
Exercise Sheet P

Issued: 09.12.2015, **Deadline:** 10.01.2016 (23:59)

General Information:

- Successful completion of the practical class requires:
  - Preparation of 40% of the exercises (there will be 30 exercises)
  - 8/20 points in the programming exercise
- The programming exercise has to be solved in groups consisting of two or three students.
- Please comment your code – points will be deducted for missing comments.
- Solutions turned in after the deadline will not be accepted.

1 Problem Statement

1.1 Project Scope

Aim of the programming exercise is the development of a chat server in Java.

The considered infrastructure consists of the mentioned server and clients. We provide an implementation of the client. Clients can register at the server and are included in a list of available clients after that. This list can be retrieved by the clients to find chat partners. Only simple text messages are allowed between clients. All communication has to be managed by the server.

For testing purposes we also provide a server implementation which gives an overview of the desired functionality.

1.2 Functionality

The chat server should provide the following functionality.

Input/Configuration

Since we are dealing with a server, hardly any user input has to be processed. Only the port number, that the server is expecting incoming connections on, should be configurable and should be passed as a command line parameter. Additionally, it should be possible to stop the server by a command.

Registration/User Administration

One of the main tasks of the server is user management. Clients should be able to log in and log out at the server and therefore, the server has to process the corresponding messages of the protocol. Additionally, the server should manage a list with all available clients, that can be retrieved by registered clients.

Message Exchange

The second main task of the server is forwarding text messages between clients. Communication between clients should be handled by the server, i.e. the server receives a text message and the recipient from a client and has to forward the message. This also includes processing messages according to the protocol.

The protocol that defines the message format for the communication between clients and server is defined in section 2.

1.3 Submission

Submit your solution via AsSESS. You can replace a submitted solution with a newer one at any time before the deadline. The last submitted solution will be evaluated.
The following files have to be submitted:

- an archive (.zip) with all *.java files
- readme.txt: Instructions how to compile (using `javac`) and run the program using the command line.
- all *.java source files of your application

1.4 Getting Started

Before you start implementing make yourself familiar with the required Java classes. Create a model that contains all elements that are necessary for the functionality of the application. Start your implementation after this model is done.

1.5 Comments and Documentation

Add comments to your source code! Points will be deducted for solutions without adequate comments. Additionally, you should document commands/command line parameters required for using your application (e.g. print them when the program starts).

1.6 Error Handling

Exceptions that might occur for example when trying to establish a connection or sending messages should be handled. Points will be deducted for an insufficient error handling.

1.7 Libraries

For your implementation only libraries from `java.*`/`javax.*` are allowed. Further libraries or code from other sources may not be used.

2 Protocol

The protocol is text based. Messages are composed of a command (represented by a single letter) and arguments. In the following **c** represents an arbitrary command. Arguments differ depending on the command.

c *argument1 [argument2 ...]*

In the following we will explain the protocol in more detail. In general, every command has to be concluded with a line break.

The server should be able to understand the following commands:

n *name* Registers a user with the name *name*. The server has to ensure that every name is only used by one user at a time. For a successful registration the server answers with an s. If the login failed the server sends an e and an error message. The *name* must not contain any blanks! Prior to a successful registration the server replies to all messages of a client with an error message. The connection between client and server should be kept up after login until logout and should be used for all communication between client and server.

m *name message* A client wants to send the message *message* to the user *name*. If the user exists, the message is forwarded by the server. Otherwise an error message is returned to the sender of the message.

t Requests the list of available users from the server. The response format is explained below.

x Ends the connection to the server. The server will not send any reply, but close the connection and delete the client from the list of available users.

The server sends the following commands:

s Acknowledges a successful login at the server (reply to **n** *name*).

m *name message* Message *message* from user *name*.
If user Alice wants to send a message with the text 'Test' to Bob, the following protocol messages are exchanged:

- Message 1 from Alice to Server: `m Bob Test`
- Message 2 from Server to Bob: `m Alice Test`

t *userlist* The **t** command is a reply to a client's **t** command. It is followed by a blank and the *userlist*. The *userlist* contains the names of all users currently logged in separated by blanks.

x *text* Announcement that the server will close the connection. The server does not expect a reply to this message, but will close the connection and delete the client from the list of available users. *text* should describe the reason for closing the connection.

e *errormsg* The server sends this message when receiving a command that cannot be processed. *errormsg* should contain details why the server could not process the command.

3 Resources

You can find two Java programs on our Website. We provide the implementation of the client that should connect to your server. Additionally, you find an example implementation of the server that you can use to get an overview of the required functionality.

3.1 Client

Get the Java program *ChatClient* from our website. It can be started with the command `java -jar ChatClient.jar <IP> <Port> [debug]`, where `<IP>` and `<Port>` are the IP address and port number of the server, respectively.
Enter `#help` to get an overview of all commands the client understands.
The client can be started in debug mode using the optional parameter *debug*. In this mode the client does not execute any commands, but all input is interpreted as a protocol message, i.e. all input is forwarded to the server without modification.

3.2 Server

Additionally, you can find an example implementation of the server on our website. Download the Java program *ChatServer* and run it with the command `java -jar ChatServer.jar [<Port>]`, where `<Port>` is the number of the port that the server listens on. Specification of the port is optional. If no port is given the server chooses a free port. The server can be stopped by pressing 'q', no further inputs are possible.

4 Hints

For your implementation you will need sockets and threads. Please refer to Learning Java (Chapters 9 and 13) or any other book about Java of your choice to learn more about using threads and sockets.

Please note:

Only turn in solutions prepared by yourself. Plagiarism will be evaluated with zero points and leads to failing the practical class!

Turn in your solutions via ASSESS (https://ess.cs.tu-dortmund.de/ASSESS/).

The programming exercise has to be solved in groups consisting of two or three students.