Der ChatServer arbeitet mit TCP als Transportprotokoll. Er verwaltet eigenständig die Verbindungen und regelt die Zustellung von Nachrichten an die Benutzer der Verbindung. Eine dezentrale Kommunikation über UDP ist somit nicht vorgesehen. Der verwendete TCP-Standardport ist 56789.

Sämtliche übermittelte Befehle werden mit UTF-8 encoded. Als Implementationshilfe gibt es dafür von Java mitgeliefert den InputStreamReader und OutputStreamWriter, die beide mit einem Encoding im Konstruktor erzeugt werden können.

Nachfolgend werden die Befehle des Protokolls aus sicht des Clienten dargestellt. Ein Request ist also das, was der Client sendet und ein Response die Antwort vom Server, die der Client empfängt.

|  |  |  |
| --- | --- | --- |
| **Request** | **Response** | **Bemerkungen** |
| auth <username> | accept | Wenn die Authentifizierung (gültiger Benutzername) erfolgt ist, können die anderen Befehle übertragen werden. |
| decline | Wenn der Benutzername ungültig ist (der Server akzeptiert Namen nach dem Schema „[a-zA-Z0-9]+“) wird ein decline verschickt und der Client muss sich erneut per „auth“-Befehl authentifizieren. |
| list\_users | users <usernames> | Der Request kann nach Bedarf vom Benutzer angefordert werden. Das Response beinhaltet die Benutzernamen der verbundenen und authentifizierten Benutzer mit Leerzeichen getrennt. |
| smsg <message> | rsmg <username> <message> | Verschickt eine Nachricht die an alle verbundenen und authentifizierten Benutzer verteilt werden soll. Das Response wird vom Server um den Benutzernamen desjenigen ergänzt, der diese Nachricht geschrieben hat. |
| quit | bye | Initiiert der Verbindungsabbau. Wird vom Server mit „bye“ quittiert. Nachdem der Server das Response schickt wird die zugehörige Verbindung (Socket) geschlossen. |

Beispiel:

C: auth L$rs

S: decline

C: auth Lars

S: accept

C: smsg Hallo

S: rmsg Lars Hallo

S: rmsg Hans Hey Lars, wie geht’s dir?

C: list\_users

S: users Lars Hans Franz

S: rmsg Franz Moin Lars

C: quit

S: bye

### Spezifikation der Klassen und Methoden

#### Klassen des Chatclients und die dazugehörigen Methoden

**ChatClient Klasse:**

Diese Klasse hat die Aufgabe den Benutzer zu authentifizieren, Sender- und Empfängerthreads zu starten und die Serveradresse, Port zu lesen und den Client zu starten.

In dem Konstruktor wird ein neuer Socket erzeugt, sowie ein Input und Outputstream für den Socket erzeugt.

*void run():* Fragt den Benutzer nach seinem Namen und liest diesen ein, ist dies erfolgreich wird der empfänger- und sender-Thread gestartet.

*boolean authenticate(username):* Hier wird geprüft, ob der Benutzer vom Server authentifiziert werden konnte. Diese Methode ist private und wird in run() benutzt.

*void main(String args[]):* Hier werden die Properties-Dateien eingelesen und der ChatClient gestartet.

**ChatClientSenderThread:**

Diese Klasse erbt von Thread und hat nur die Methode run() implementiert. Diese Methode wartet auf Eingabe von dem Benutzer und liest die Nachrichten ein. Sollte die Nachricht „/quit“ lauten, wird quit in den outputstream gegeben. Außerdem gibt es noch den Befehl „/list-users“, welcher die Teilnehmer Liste des Chats ausdruckt.

**ChatClientEmpfaengerThread**:

Diese Klasse erbt ebenfalls von Thread und implementiert nur die Methode run(). Jede eingegangene Nachricht wird zuerst geprüft, welchem Response sie angehört. Bei „rmsg“ wird die Nachricht und der Benutzer der sie geschickt hat angezeigt, „users“ ist die Antwort des Servers auf den „list\_users“-Befehl und zeigt die verbundenen Clienten an und „bye“ dient der Quittierung des „quit“-Befehls und veranlasst die Beendung des ClientEmpfaengerThreads.

**ChatClientUI:**

DieseKlasse ist ein Interface und wird von ChatClientCLI implementiert. Sie dient der Kommunikation des Benutzers mit dem ChatClient (Eingabe / Ausgabe).

*String getEingabe()*

*void showStatusmeldung(String meldung)*

*void showUsers(List<String> users)*

*void showMessage(String username, String message)*

**ChatClientCLI:**

Implementation von ChatClientUI.

*String getEingabe():* Liest die nächste Eingabe und gibt diese wieder

*void showStatusmeldung(String meldung):* Stellt Meldungen dar, die vom ChatClient selbst erzeugt wurden

*void showUsers(List<String> users*): Zeigt die Liste der User an

*void showMessage(String username, String message):* Gibt die eingegebene Nachricht aus und gibt sie in der Form(Name: Nachricht aus)

#### Klassen des Chatservers und die dazugehörigen Methoden

**ChatServer**

Der ChatServer ist der zentrale Part des Servers. Er verwaltet die anderen Teile wie den ChatServerSocket und die ChatServerWorkerThreads. Außerdem startet er einen ServerEingabeThread.

Er stellt über die Methode „getConnectedWorkers()“ alle Verbindungen zur Verfügung, die bereits mit einem Benutzernamen authentifiziert wurden und sich quasi in dem Chatraum befinden. Mittels „sendAll(username:String, message:String)“ als Convenience-Method können Nachrichten an alle anderen Benutzer geschickt werden.

Die Methoden acquire() und release(worker:ChatServerWorkerThread) dienen dem Semaphore-Mechanismus. Während „acquire“ vom ChatServerSocket aufgerufen wird, um das weitere Akzeptieren von Verbindungen zu unterbinden, sollte das Limit erreicht sein, wird „release“ vom ChatServerWorkerThread selbst aufgerufen, wenn dieser geschlossen wird um wieder einen Platz frei zu machen und als verbundenen Benutzer am ChatServer auszutragen. „addWorker(worker:ChatServerWorkerThread)“ dient dementsprechend dem Beitreten zum Chatserver und wird vom Worker aufgerufen, sobald die Authentifizierung abgeschlossen ist (vorher ist der Benutzer noch nicht im Chat).

Zuletzt gibt es eine „run()“-Methode, die den Server und seine Threads startet.

**ChatServerSocket**

Der ChatServerSocket kümmert sich um die eingehenden Verbindungen, erzeugt einen ChatServerWorkerThread dafür und startet diesen. Nach einiger Überlegung hätte hier auch die Authentifizierung geschehen können, damit die Registrierung des Workers am ChatServer ebenfalls hier geschehen kann. Das würde jedoch bedeuten, dass der ChatServerSocket solange blockiert ist, bis der Benutzer der eingegangenen Verbindung authentifiziert ist.

Der ChatServerSocket ist als Thread realisiert, der mit den Methoden start() und stop() gestartet und gestoppt werden kann. Die run-Methode sollte wie für Threads üblich nicht selbst aufgerufen werden.

**ChatServerWorkerThread**

Kümmert sich um die Verbindung mit dem Client und die Abhandlung des Protokolls. Er ist als Thread realisiert. Deswegen kann er sich eigenständig am ChatServer registrieren (addWorker(…)) und abmelden (release(…)). Außerdem bietet er über die „send(message:String)“-Methode an, eine Nachricht an den Clienten der hinter der Verbindung steht zu schicken.

Über eine private Methode checkUsername(username:String) kann geprüft werden, ob der vom Clienten gewählte Benutzername erlaubt ist. Derzeit kann ein Benutzername aus mindestens einem oder beliebig vielen Buchstaben oder Zahlen bestehen. Die Methode könnte jedoch um weitere Prüfungen erweitert werden (z.B. keine doppelten Namen, etc.).

**ServerEingabeThread**

Wird für die administrative Eingabe auf dem Server benutzt. Ein Admin könnte mit bestimmten Befehlen so Funktionen auf dem ChatServer aufrufen. Derzeit wird nur ein „quit“-Befehl unterstützt der den ChatServerSocket stoppen soll, damit keine weiteren Verbindungen mehr angenommen werden können, die verbundenen Benutzer aber noch weiter schreiben können. Leider funktioniert dies noch nicht in der gewünschten Qualität. Wie der Name bereits verrät handelt es sich bei dieser Klasse um einen Thread der eigenständig läuft.