SPEZIFIKATION EINES CHAT PROTOKOLLS

Amanpreet Kaur & Till Poland

MAY 22, 2017

RECHNERNETZE PRAKTIKUM – AUFGABEN 2 & 3

Inhaltsverzeichnis

1. Systemüberblick

Die nachfolgende Spezifikation beschreibt eine Client-Server Architektur, die zum Austausch von Nachrichten zwischen verschiedenen Benutzern bentuzt werden soll. Sie setzt sich aus verschiedenen Klassen zusammen:

- ChatClient
- ChatServer
- ClientThread
- ChatRoom
- ChatMessage

1.1 ChatClient

Die Komponente ChatClient kann sich als Client mit dem Server unterhalten. Von ihm aus ist es dem Benutzer möglich, sich am Server und darauf bei Chatrooms anzumelden und mit anderen Clients zu chatten.

1.2 ChatServer

Der ChatServer lauscht auf einkommende Verbindungen von Clients und verarbeitet Nachrichten und befehle von verbundenen Clients.

Außerdem verwalter er die Chatrooms und das Versenden von Nachrichten an andere Benutzer.

1.3 ClientThread

Der ClientThread wird erstellt, sobald sich ein ChatClient beim Server anmeldet und stellt den Client auf Serverseite dar.

1.4 ChatRoom

Die ChatRooms sind Räume auf dem Server, an denen sich Benutzer anmelden können um mit anderen darin exklusiv zu chatten.

1.5 ChatMessage

Die Nachrichten, die der Client dem Server sendet, sind ChatMessages, die von diesem interpretiert werden. Sie ermöglichen die Interpretation von Befehlen auf dem Server vom Client.

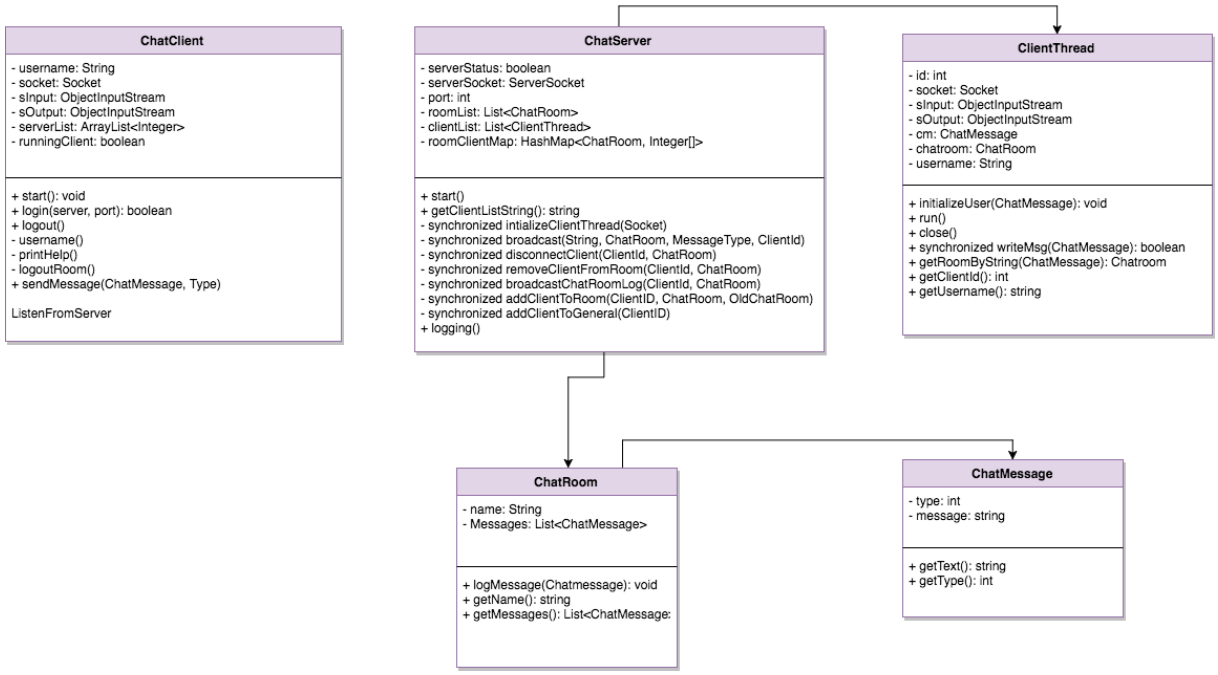
2. Technische Aspekte

Unter diesem Punkt werden die technischen Aspekte des Servers und Clients aufgelistet.

2.1 System-Überblick

Dieses UML Diagramm beschreibt die Klassen und Funktionen einer eigenen Implementation von dem Client und den Aufbau des ChatServers.

Sie dient nur zur Verbildlichung des Aufbaus.



2.2 Server

Der Server ist ein auf Java 1.8 basierender Server, der die Clients als einzelne Threads verwaltet.

Er hört kontinuierlich auf Socketanfragen auf seine IP und seinen Port. Er versteht allerdings nur eine bestimmte Sorte an Objekten.

2.3 ChatMessage

Die ChatMessage ist wie oben beschrieben eine eigene Klasse und der Server ist so implementiert, dass er nur Nachrichten vom Typ ChatMessage akzeptiert bzw. er castet ankommende Objekte zu einer ChatMessage. Falls dies fehlschlägt wird die Nachricht nicht verstanden.

Die ChatMessage hat eine Nachricht (String) und einen Typ (Integer) als Parameter. Die ChatMessages sind über einen Typ mit einem bestimmten Ereignis verbunden. Wenn der Client eine Nachricht an den Server bzw. Chatroom verschicken möchte muss diese Nachricht vom Typ ChatMessage<String, Int> sein.

Erforderliche Funktionen die der Server aufruft sind:

- getText() : Gibt den String der ChatMessage zurück.
- getType(): Gibt den Typen der ChatMessage zurück.

Außerdem muss die ChatMessage Serializable sein, da dies von dem Server vorrausgesetzt wird.

Da der Server über Java läuft, sind einige diesbezügliche Dinge durch Java festgelegt: Die Implementierung von java.io.Serializable und das Senden über einen ObjectOutputStream, über den Java Objekte an den Server gesendet werden.

2.2 Port

Der von uns benutzte Port ist der Port 50 000.

Dieser ist nach der Liste der IANA noch nicht belegt.

Quelle: https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

2.3 Protokoll

Vom Client aus werden ChatMessage Objekte über einen Socketstream über eine TCP Verbindung gesendet, welche in den technischen Details spezifiziert sind.

2.3.1 Anmeldung von Chatclient

Zur Anmeldung am Server baut der Client eine Verbindung mit dem Server über ein Socket auf. Die Socketverbindung verläuft über die IP des gestarteten Servers (zu Testzwecken z.B. localhost) und über den oben spezifizierten Port.

a. Darauf folgend, muss eine Initialisierungsnachricht gesendet werden, die folgendermaßen aufgebaut ist:

   Text: Der gewählte Benutzername

   Befehl / ChatMessage Typ: Initialize

   Info: Zwei unterschiedliche Benutzer können den gleichen Benutzernamen haben!

   Danach befindet sich der Client im generellen Chatroom „General“. Von diesem aus werden Nachrichten nicht an andere Benutzer gesendet. Alle Befehle sind allerdings möglich.

b. Weitere Kommunikation

   Nach der Initialisierung nimmt der Server alle der unter “Technische Details” spezifizierten Befehle entgegen und beantwortet diese sinngemäß. Dieser Austausch findet über dieselbe Socketverbindung statt, mit der auch die Initialisierung stattgefunden hat.

   Geschrieben werden kann anderen Benutzern nur über neue ChatRooms, nicht über den generellen.

   Alles, was der Server an den Client sendet wird als String gesendet. Darunter fallen Antworten auf Befehle, Nachrichten von anderen in einem ChatRoom und Fehlermeldungen.

   Wird diese Verbindung unterbrochen, muss sich der Client auf dem Server neu initialisieren.

2.3.2 Anmeldung an einem ChatRoom

Der ChatRoom hat einen eindeutigen Namen, über den dieser angesprochen bzw. erreicht werden kann. In einem Chatroom können sich mehrere Clients unterhalten, sprich sie verschicken Nachrichten untereinander.

Außerdem verwaltet jeder Chatroom eine Nachrichtenliste bzw. ein Nachrichtenlog, mit den Nachrichten die in diesem Raum versendet wurden. Sprich der Raum führt ein Protokoll über die gesendeten Nachrichten und gibt dieses auch an neue Benutzer im Raum an.

Die Anmeldung an einem ChatRoom findet nach der Anmeldung am Server über den „Join“ Befehl statt.

2.3.3 Timeout

Der Benutzer wird nach 5 Minuten, in der er/sie nichts geschrieben hat, automatisch abgemeldet und vom Server gelöscht.

2.3.4 Fehlermeldungen

Fehlermeldungen werden wie auch alles andere vom Server als Strings an den Client gesendet, behindern aber nicht den weiteren Verlauf des Chats. Sie sind bei den Befehlen in den Technischen Details aufgelistet.

2.3.5 Textkodierung

Die Nachrichten müssen in UTF-16 kodiert sein, da dies der nativen Stringkodierung von Java entspricht. Andernfalls kann der Server diese gegebenenfalls nicht korrekt darstellen bzw initialisieren.

3 Technische Details

In dem folgenden Abschnitt werden die Technischen Daten unseres Systems beschrieben und zusammengefasst.

3.2 Chatrooms

Falls sich der letzte Benutzer aus einem Chatroom ausloggt, wird der Chatroom direkt gelöscht.

3.3 Befehle / ChatMessage Typen

Die Befehle, die auf dem Server ausgeführt werden, werden über einen Integer Wert typisiert. Je nachdem, welcher Integer gesendet wird, wird die gesendete Nachricht anders vom Server behandelt.

Die Antwort vom Server ist immer ein String. Auch wenn es eine Fehlermeldung ist.

Die Liste der Typen sieht folgendermaßen aus:

Logout - Wert: 0

Folge:	Abmeldung und Löschung von dem aufrufenden Benutzer auf dem Server.
Antwort d. Servers:	Keine Antwort.

Message - Wert: 1

Folge:	Die Nachricht wird an alle anderen Benutzer im momentanen ChatRoom gesendet, außer der Benutzer ist im generellen ChatRoom.
Antwort d. Servers:	Keine Antwort.

List Users - Wert: 2

Folge:	Der Server gibt eine Liste aller momentan angemeldeten Benutzer heraus, aufgelistet mit usernamen.
Antwort d. Servers:	Liste aller angemeldeten Benutzer

Join Chatroom - Wert: 3

Folge:	Der Benutzer wird aufgefordert, den Namen des beizutretenden ChatRooms einzugeben.
Antwort d. Servers:	Log des ChatRooms
Fehler:	ChatRoom nicht vorhanden

List Chatrooms – Wert 4

Folge:	Der Server listet alle ChatRooms auf, die auf dem Server existieren
Antwort d. Servers:	Liste aller ChatRooms, die auf dem Server gespeichert sind.

Chatroom logout- Wert 5

Folge: Der Benutzer wird aus dem momentanen Chatroom gelöscht und in den generellen ChatRoom verschoben.

Falls der Benutzer schon im generellen Chatroom war, passiert nichts.

Antwort d. Servers: Keine Antwort

Create Chatroom – Wert 6

Folge:

1. Von dem Benutzer wird ein Name abgefragt (Antwort d. Servers), der als Name für den Server benutzt werden soll.
2. Nach der Eingabe eines Namens wird der ChatRoom mit dem gewählten Namen erstellt und der Benutzer dem ChatRoom hinzugefügt.

Antwort d. Servers: Keine Antwort

In Chatroom – Wert 8

Folge: Der Server gibt den Namen des Chatrooms zurück, in dem sich der Benutzer momentan befindet.

Antwort d. Servers: Momentaner Chatroom

Users in chatroom – Wert 9

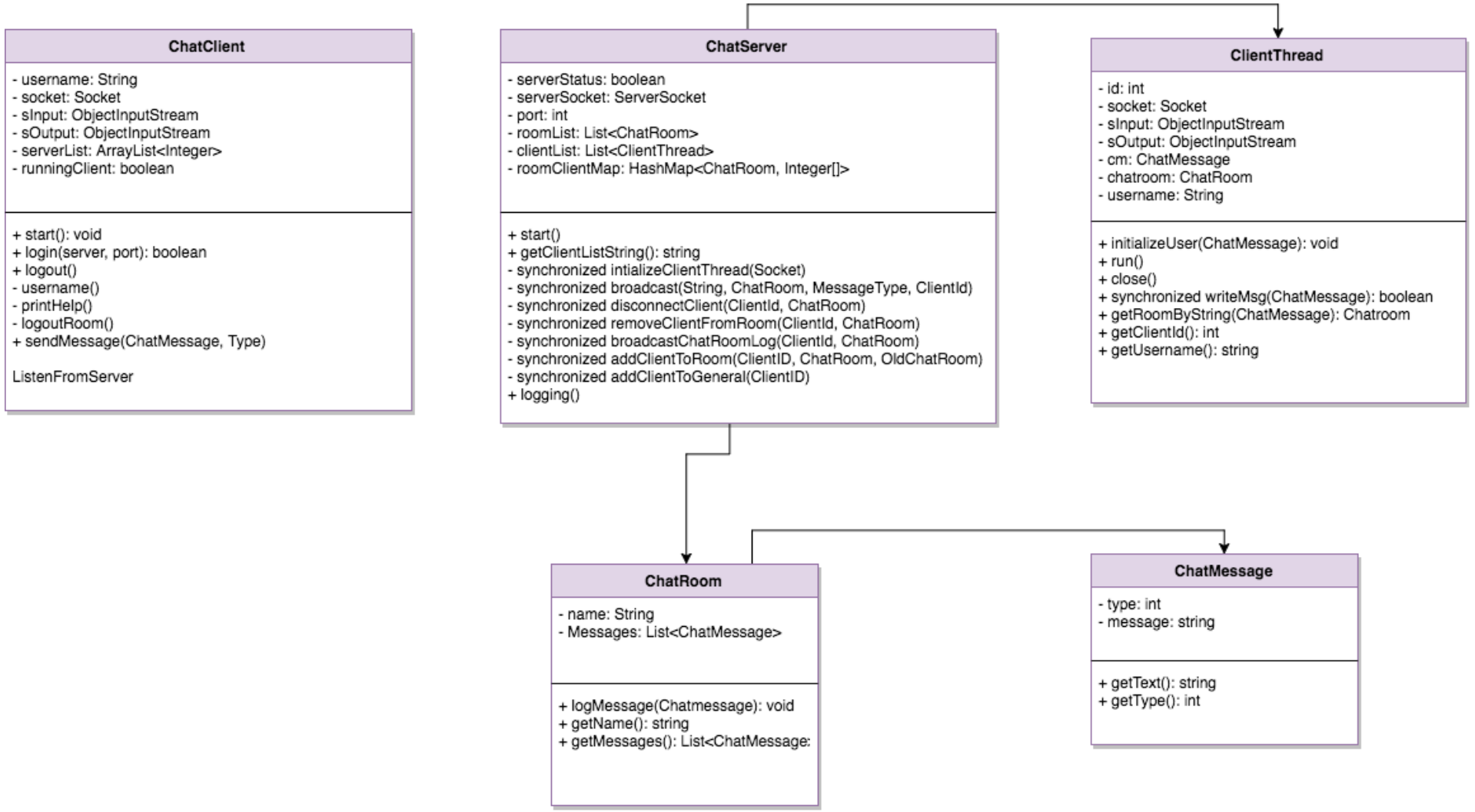
Folge: Der Server gibt als String eine Liste aller Benutzer zurück, die momentan in dem Chatroom sind, in dem der Benutzer sich befindet.

Antwort. d. Servers: Liste aller Benutzer, die in dem Chatroom sind, in dem der aufrufende Benutzer sich befindet.

Initialize – Wert 11

Folge: Wird zur Initialisierung am Server verwendet. Der Benutzer ist mit dem mitgegebenen Benutzernamen initialisiert.

Antwort d. Servers: Keine Antwort


ChatClient
- username: String
- socket: Socket
- sInput: ObjectInputStream
- sOutput: ObjectInputStream
- serverList: ArrayList<Integer>
- runningClient: boolean
+ start(): void
+ login(server, port): boolean
+ logout()
- username()
- printHelp()
- logoutRoom()
+ sendMessage(ChatMessage, Type)
ListenFromServer
ChatServer
- serverStatus: boolean
- serverSocket: ServerSocket
- port: int
- roomList: List<ChatRoom>
- clientList: List<ClientThread>
- roomClientMap: HashMap<ChatRoom, Integer[]>
+ start()
+ getClientListString(): string
- synchronized intializeClientThread(Socket)
- synchronized broadcast(String, ChatRoom, MessageType, ClientId)
- synchronized disconnectClient(ClientId, ChatRoom)
- synchronized removeClientFromRoom(ClientId, ChatRoom)
- synchronized broadcastChatRoomLog(ClientId, ChatRoom)
- synchronized addClientToRoom(ClientID, ChatRoom, OldChatRoom)
- synchronized addClientToGeneral(ClientID)
+ logging()
ClientThread
- id: int
- socket: Socket
- sInput: ObjectInputStream
- sOutput: ObjectInputStream
- cm: ChatMessage
- chatroom: ChatRoom
- username: String
+ initializeUser(ChatMessage): void
+ run()
+ close()
+ synchronized writeMsg(ChatMessage): boolean
+ getRoomByString(ChatMessage): Chatroom
+ getClientId(): int
+ getUsername(): string
ChatRoom
- name: String
- Messages: List<ChatMessage>
+ logMessage(Chatmessage): void
+ getName(): string
+ getMessages(): List<ChatMessage:
ChatMessage
- type: int
- message: string
+ getText(): string
+ getType(): int