Hochschule Kaiserslautern

Entwicklung verteilter Systeme

Multi-user Chat mit Java Socket

Ujwal Subedi
Wissam Alamareen

unter Aufsicht von
Prof. Dr. Manuel. Duque-Antón

4. Februar 2021

Inhaltsverzeichnis

1 Einleitung

Die Socket-Programmierung wird zum Entwickeln von Client-Server-Anwendungen in Java verwendet. In der Client-Server-Architektur stellt der Server einen Dienst bereit, und der Client kann diesen Dienst verwenden, um die gewünschte Ausgabe oder das gewünschte Ergebnis zu erhalten. Mit Sockets können auch zwei oder mehr Rechner über das Netzwerk mithilfe des TCP / IP-Protokolls miteinander kommunizieren. In diesem Projekt man lernt, wie Client-Server-Anwendungen in Java mithilfe der Socket- Programmierung geschrieben wird. Außerdem erfahrt man, wie eine Chat-Anwendung für mehrere Benutzer / Gruppen in Java mit Chat-Protokollierungsfunktion geschrieben wird. Alle Konzepte werden in einer einfachen Sprache erklärt. Neben der Socket-Programmierung bietet dieser Projekt auch eine kurze Einführung in einige der Java-Konzepte, die zum Verständnis unserer Chat-Anwendung erforderlich sind. Diese Chat-Anwendung vermittelt Ihnen ein konkretes Verständnis Grundlagen der Socket-Programmierung.

2 Implementierte Funktionen

2.1 Messaging

Eine Chat-Anwendung sollte sowohl das Senden als auch das Empfangen ermöglichen und gleichzeitig verarbeiten. Dies wird in dieser Anwendung mit dem Java-Multithreading-Konzept erreicht.

2.2 Group Chat

Eine weitere wichtige Funktion in der Chat-Anwendung ist der Gruppenchat, der in dieser Anwendung implementiert ist. Es ermöglicht Menschen zu chatten. Die Nachricht wird zusammen mit dem Namen des Benutzers, der die Nachricht gesendet hat, an alle Benutzer im Chatroom, indem alle sich befinden, gesendet. Benutzer, die im Chatroom verfügbar sind, erhalten die Nachricht.

2.3 Nachrichten Form

- Direct and Group Messaging

Nachdem der Benutzer den Benutzername und das Passwort zum Login durch UI Login Fenster eingegeben habe, werden diese an den Server gesendet. In Server über-

prüft eine Methode Login, in dem sie der Benutzername und das Passwort mit den vorhandenen Benutzer vergleicht, wenn der aktuelle Benutzer in Server schon gespeichert ist, ist sein Login erfolgreich abgeschlossen. Nachher ist ihm möglich die online Benutzer zu sehen. login <user> <password>

- Start chatting

Jeder Benutzer hat die Möglichkeit mit den online Benutzer zu Kommunizieren, wenn er einen zum kontaktieren auswählt ,ihm Nachricht schreibt und senden Button drückt, wird die Nachricht mit dem Benutzername an den Server im folgendem Form weitergeleitet Msg <user> <msg-body>.
der Server empfängt die Nachricht und sendet sie erneut zu dem Benutzer, dessen Benutzername in der selben Nachricht steht

- join Lobby

Der Client drückt auf den Button Join dann gibt den Namen der Gruppe danach wird der Folgenden Form an Server gesendet
join <#Gruppenname> Server fügt den neuen Client in die Lobby hinzu.

- Send Message zu einer Lobby

Client wählt eine Lobby aud und schreibt eine Nachricht.Die Nachricht wir in der folgenden From gsendet an den Server
msg <Gruppenname> <msg-body>
Server sendet dann die Nachricht <msg-body> an alle Mitglieder dieser Gruppe

3 Socket

Die Java Socket-Programmierung wird für die Kommunikation zwischen Anwendungen verwendet, die auf verschiedenenJRE ausgeführt werden. Im Allgemeinen gibt es zwei Arten der Netzwerkkommunikation: TCP (Transport Control Protocol) und UDP (User Datagram Protocol). TCP und UDP werden für unterschiedliche Zwecke verwendet und haben beide eindeutige Einschränkungen: [3]

3.1 TCP

TCP ist ein relativ einfaches und zuverlässiges Protokoll, mit dem ein Client eine Verbindung zu einem Server herstellen und ,4 die beiden Systeme kommunizieren können. In TCP weiß jede Entität, dass ihre Kommunikationsnutzdaten empfangen wurden.

3.2 UDP

UDP ist ein verbindungsloses Protokoll und eignet sich für Szenarien, in denen nicht unbedingt jedes Paket am Ziel ankommen muss, z. B. Medien-Streaming. Der Client in der Socket-Programmierung muss zwei Informationen kennen: IP-Adresse des Servers und Port-Nummer. Der folgende Code öffnet eine Verbindung zu einem Server:

Client client = new Client("localhost", 4444);

Der Client versucht, über diese Portnummer und IP-Adresse eine Verbindung zum Server herzustellen. Der Server hört zu und akzeptiert die Verbindung. Sobald die Verbindung hergestellt ist, kann der Server eine Nachricht vom Client empfangen(Eingabestreams) und eine Nachricht an den Client(Ausgabestreams)zurücksenden.

Da das Projekt mehrere Clients umfasst, die Nachrichten aneinander senden können, werden Threads verwendet Sehe den folgenden Code.

```
@Override
    public void run() {
        try {
            ServerSocket serverSocket = new ServerSocket(port);
            while (true) {
                mSocket = serverSocket.accept();
                System.out.println("Connection : " + mSocket);
                ChatServer chatServer = new ChatServer(this, mSocket);
                chatServers.add(chatServer);
                clientProcessorService.submit(chatServer);
                System.out.println("Anzahl Clients: " + chatServers.size());
            }
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
```

Für die Serveranwendung verwenden wir die Servermanager klasse, die an eine bestimmte Portnummer gebunden ist. Wir erstellen dann einen Server-Socket und warten auf eine Verbindung mit dem Client und akzeptieren diese.

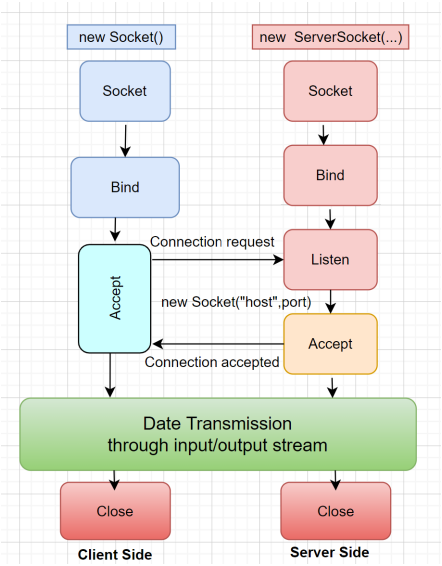


Abbildung 1: Socket's life cycle (Java).

4 Architektur

Diese Anwendung basierend auf Client / Server implementiert Modell.

4.1 Server

Ein Server kann ein Computer sein, auf dem ein Server ausgeführt wird Diese Servercomputer akzeptieren Clients über Netzwerkverbindungen, die angefordert werden. Der Server antwortet, indem er angeforderte Antworten sendet. Es gibt viele verschiedene Serveranwendungen, die je nach dedizierter Arbeit variieren. Einige sind für die Annahme von Anforderungen und die Ausführung aller dedizierten Arbeiten wie Geschäftsanwendungsserver beteiligt

4.2 Client

Ein Client ist ein Software-Anwendungscode oder ein System, das eine andere Anwendung anfordert, die auf einem dedizierten Rechner (Server genannt) läuft. Diese Clients müssen nicht mit dem Server über eine drahtgebundene (wired)Kommunikation verbunden sein. Die drahtlose (wireless) Kommunikation findet in diesem Prozess statt. Ein Client mit einer Netzwerkverbindung kann eine Anfrage an den Server senden. Zuerst muss der Server gestartet werden, um den Socket an den lokalen Host und den dedizierten Port zu binden.Nach dem Start des Servers beginnt er, die Client-Anfrage zu akzeptieren und die Kommunikation findet statt.

4.3 Nachrichtentausch

5 Multithreading

Mit der Entwicklung des Prozessorkerns wurde es wichtig, ein Programm zu schreiben, das den Computerkern nutzt, um die Effizienz zu erhöhen. Multithreading ist eine Art von Ausführungsmodell, bei dem mehrere Threads im Kontext eines Prozesses existieren können, so dass sie unabhängig voneinander ausgeführt werden, aber ihre Prozessressourcen gemeinsam nutzen. [2] Für eine Chat-Anwendung sollte ein Server

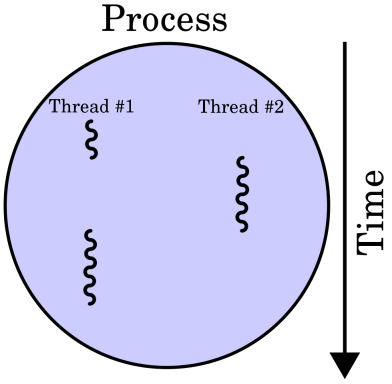


Abbildung 2: Ein Prozess mit zwei Ausführungssträngen, der auf einem einzigen Prozessor läuft.

[4]

viele Clients verarbeiten, was bedeutet, dass ein Client von einem einzelnen Thread verarbeitet werden muss. Mit dieser Lösung entsteht ein weiteres Problem, das zu berücksichtigen ist, nämlich die Kapazität der Serverarchitektur. Wie viele Threads kann ein Server verarbeiten, nicht wie viele Clients. Um dieses Problem zu lösen, haben wir uns das Thread-Pooling ausgedacht.

5.1 Thread Pooling

Der Thread-Pool, der auch als replizierter Worker oder Worker-Crew-Modell bekannt ist, ermöglicht die gleichzeitige Ausführung mehrerer Threads und das Warten auf die Zuteilung bestimmter Aufgaben zur Ausführung durch solche erhöht die Leistung des Programms und vermeidet Latenz in der Laufzeit aufgrund der häufigen Erstellung und Zerstörung von Threads für kurzlebige Aufgaben.

```
1 //Beispiel ThreadPoolExecutor
2     private ExecutorService clientProcessorService = new ThreadPoolExecutor(10,
    25, 0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());
```

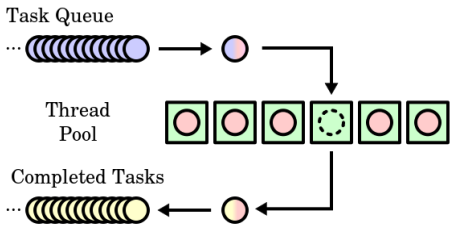


Abbildung 3: Thread Pooling
[5]

In Java werden Threads auf Systemebene abgebildet, bei denen es sich um Ressourcen des Betriebssystems handelt. Wenn Sie unkontrolliert Threads erzeugen, können Ihnen diese Ressourcen schnell ausgehen. Die Kontextumschaltung zwischen Threads wird ebenfalls vom Betriebssystem vorgenommen - um Parallelität zu emulieren. Vereinfacht betrachtet gilt: Je mehr Threads Sie erzeugen, desto weniger Zeit verbringt jeder Thread mit der eigentlichen Arbeit. Das Thread-Pool-Muster hilft, in einer Multithread-Anwendung Ressourcen zu sparen und auch die Parallelität in bestimmten vordefinierten Grenzen zu halten. [1]

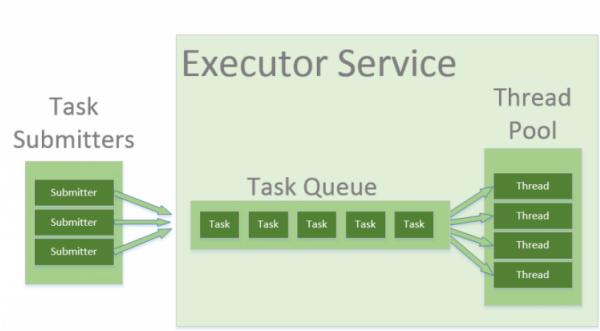


Abbildung 4: Thread Pooling
[1]

6 UML(Unified Modeling Language) Diagram

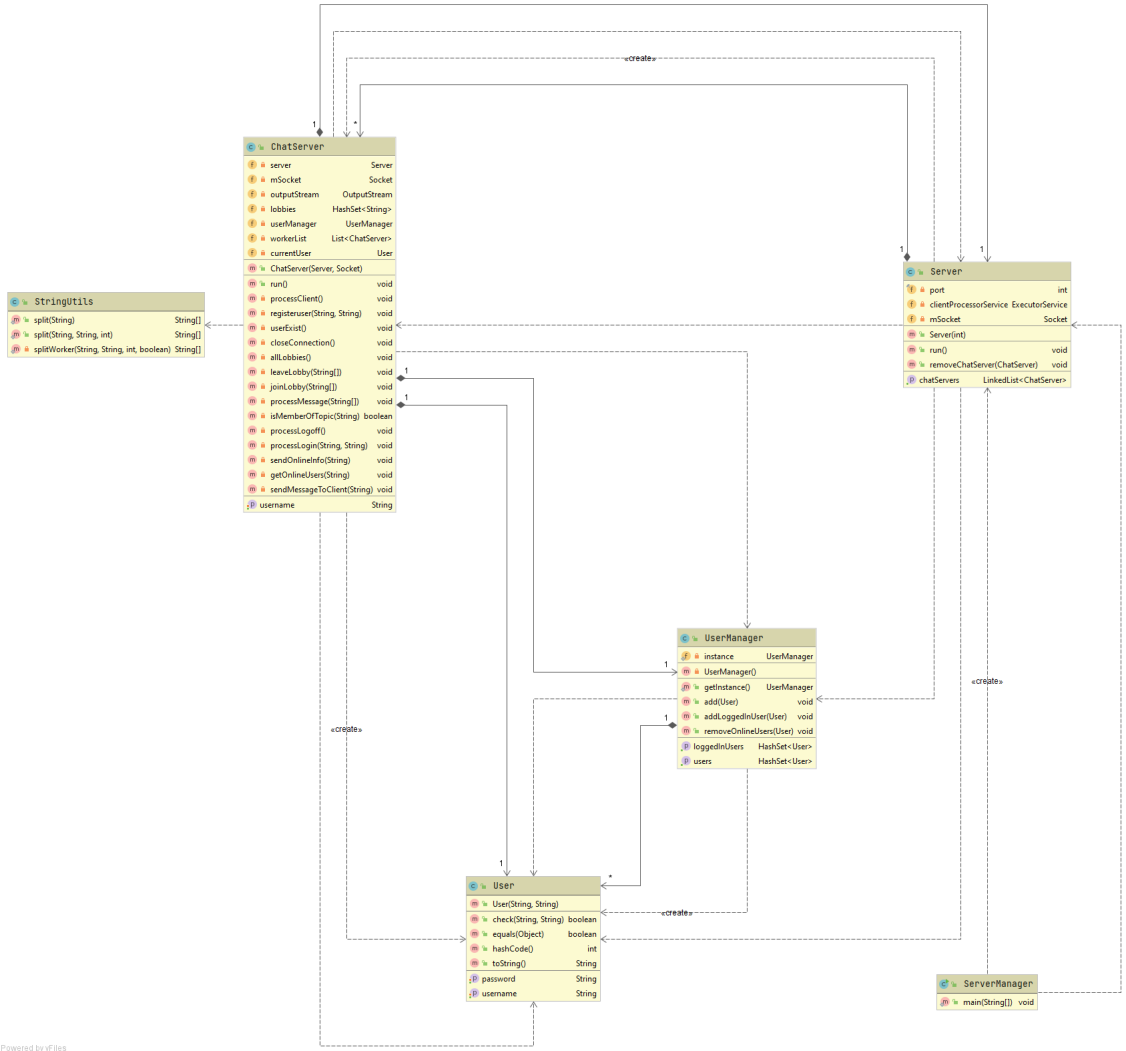


Abbildung 5: Server UML

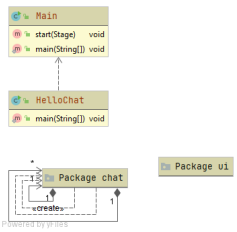


Abbildung 6: Client 1

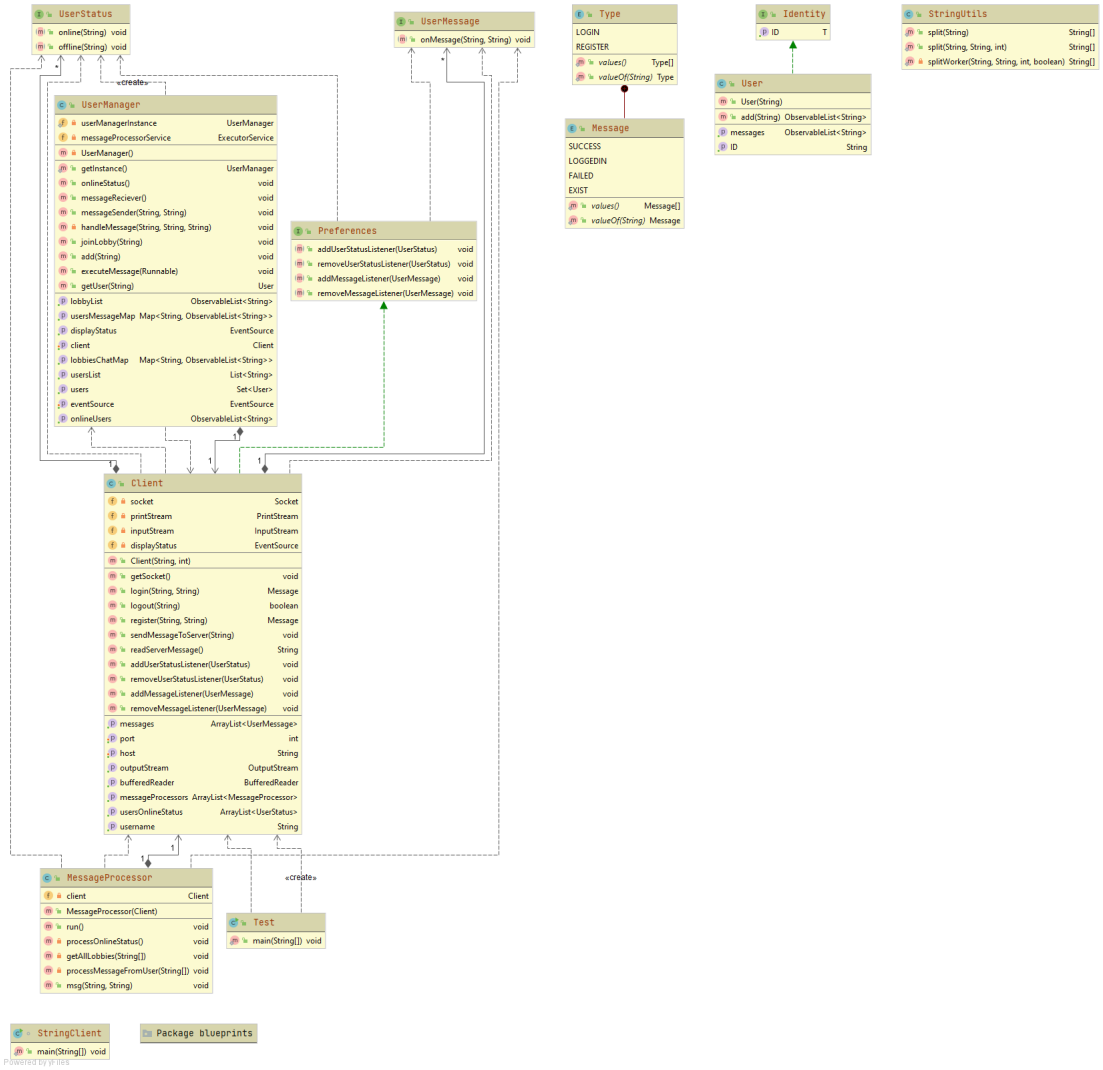


Abbildung 7: Client 2

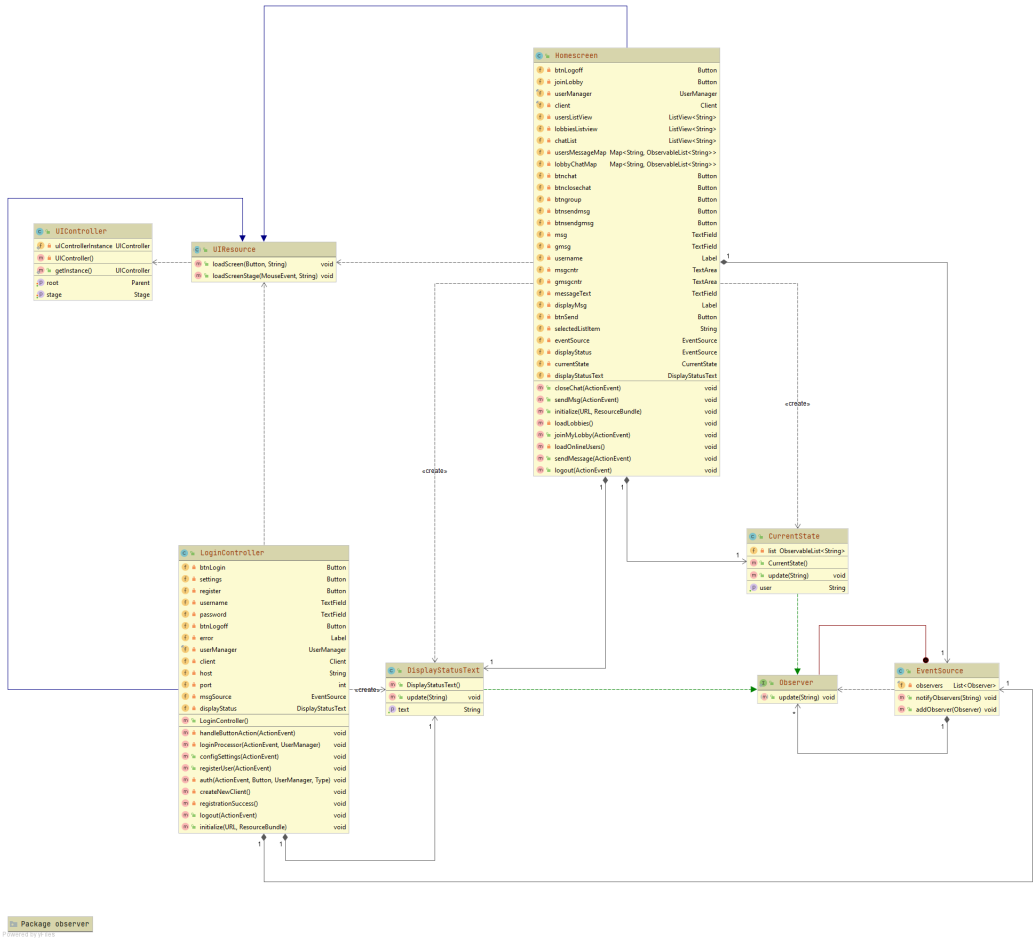


Abbildung 8: Client 3

7 Program Test

Unser Client-Programm ist so konzipiert, dass wir versucht haben, einige Probleme zu lösen, auf die wir gestoßen sind. Einige der Tests sind im Folgenden beschrieben:

7.0.1 Username/Passwort falsch

7.0.2 User schon eingeloggt

7.0.3 Keine Connection mit dem Server

8 User Guides

8.1 Einleitung

HelloChat ist eine Chat-Anwendung, in der Sie mit allen Online-Benutzern eines bestimmten Servers chatten können. Sie können auch einer öffentlichen Lobby beitreten, in der Sie mit allen Benutzern chatten können, die der Lobby beigetreten sind.

8.2 Anmelden/Registrieren

HelloChat ermöglicht es Ihnen, einen neuen Benutzer anzulegen, damit Sie mit anderen Online-Benutzern chatten können.

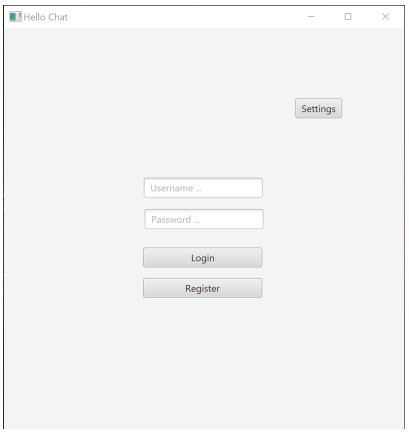


Abbildung 9: Login Fenster

8.3 Einstellung

In der Einstellung können Sie die Server-IP-Adresse und den Port ändern.

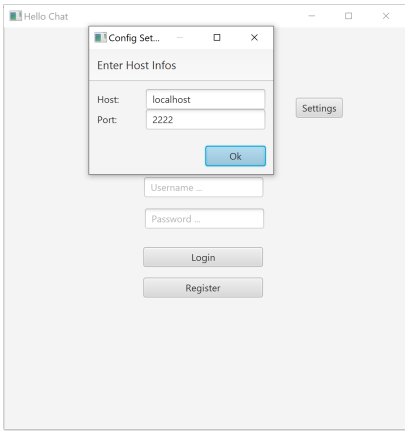


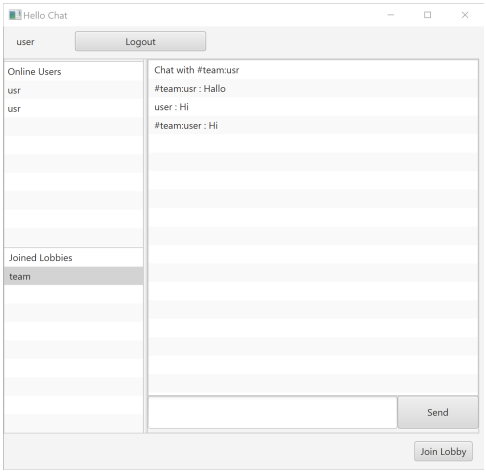
Abbildung 10: Setting Port and Host

8.4 Benutzer-zu-Benutzer Chat

Im Bereich Online-Benutzer sehen Sie alle Online-Nutzer, die auf dem Server angemeldet sind. In dieser Anwendung dürfen Benutzer mit allen Online-Benutzern des Servers chatten.

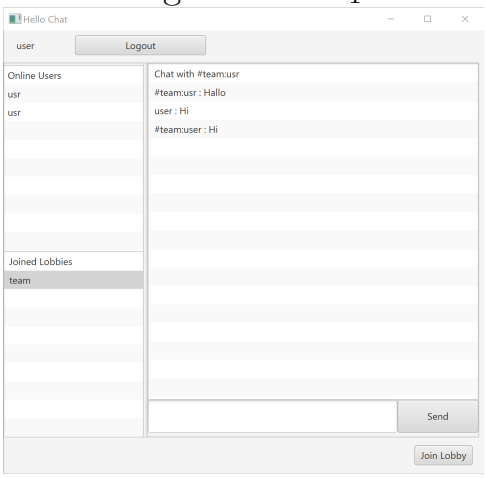
8.5 Benutzer zu Lobby Chat

Sie sehen eine Liste der Lobbys, denen Sie beigetreten sind, und dürfen in der ausgewählten Lobby eine Nachricht schreiben. Alle Benutzer, die der Lobby beigetreten sind, können Ihre Nachricht sehen.



[b]0.4

Abbildung 11: Client$_1$.



[b]0.4

Abbildung 12: Client$_2$.

Abbildung 13: lobby chat .

8.6 Ausloggen

Wenn man den Chat beenden möchte, sollte nur auf dem Button Logout geklickt werden und nachher hat man die Möglichkeit jederzeit wieder einzuloggen, weil die Daten von Benutzer gespeichert sind. Wenn jemand das Fenster schließt, meldet er sich nicht ab, sondern schließt das Programm, so dass derselbe Benutzer bei einem erneuten Anmeldeversuch feststellt, dass er bereits angemeldet ist. In diesem Szenario wird die Abmeldeschaltfläche im Anmeldefenster aktiviert.

Zusammenfassung

Client-Server-Programmierung in Java ist einfach. Das Paket java.net hat eine sehr starke und flexible Infrastruktur für die Netzprogrammierung zur Verfügung gestellt. In diesem Projekt haben wir resümiert, wie Sie die Standard-Java-Netzwerk- und I / O-Klassen verwenden und wie Sie gleichzeitige I / O sicher mithilfe von Threads behandeln. Wir erstellen eine Chat-GUI und erweitern die aktuellen Programme um identifizierbare Benutzer und Benutzergruppen Die Chat-Anwendung bietet ein besseres und flexibleres System für chatten. Es wurde mit den neuesten fortschrittlichen Technologien entwickelt, um ein zuverlässiges System bereitzustellen. Diese Anwendung kann einen besseren Bedarf finden, wenn zusätzliche Funktionen beispielweise Voice-Chat im System verfügbar sein werden.

Literatur

[1] Eugen Paraschiv. *Introduction to Thread Pools*. URL: https://www.baeldung.com/thread-pool-java-and-guava. (accessed: 04.07.2020).

[2] Techopedia. *Multithreading*. URL: https://www.techopedia.com/definition/24297/multithreading-computer-architecture. (accessed: 04.07.2020).

[3] uni-weimar. *Socket Tcp*. URL: https://www.uni-weimar.de/kunst-und-gestaltung/wiki/TCP/IP_UDP. (accessed: 04.07.2020).

[4] Wikipedia. *Thread Pool*. URL: https://en.wikipedia.org/wiki/Multithreading_(computer_architecture). (accessed: 04.07.2020).

[5] Wikipedia. *Thread Pool*. URL: https://en.wikipedia.org/wiki/Thread_pool. (accessed: 04.07.2020).