Dokumentation Client Server Programm

Inhalt

[Grundprinzip 1](#_Toc21443193)

[Hauptserver 2](#_Toc21443194)

[Codeerklärung 2](#_Toc21443195)

[Serversession 2](#_Toc21443196)

[Konstruktor 3](#_Toc21443197)

[Run 3](#_Toc21443198)

[ExecuteCommand 4](#_Toc21443199)

[ExecuteOnSubserver 4](#_Toc21443200)

[Chat Methode 5](#_Toc21443201)

[Subserver / Subserver2 6](#_Toc21443202)

[Main 6](#_Toc21443203)

[Execute\_command 6](#_Toc21443204)

[Client 7](#_Toc21443205)

[Main 7](#_Toc21443206)

[UDP\_Client 8](#_Toc21443207)

[Konstruktor 8](#_Toc21443208)

[Run 8](#_Toc21443209)

[Zusatzinformationen 8](#_Toc21443210)

[Verbindungsaufbau: Server – andere Geräte (TCP) 8](#_Toc21443211)

[Dateien zwischen Clients schicken (UDP) 9](#_Toc21443212)

[Technische Umsetzung 9](#_Toc21443213)

[Persönliche Meinung/Anmerkung 9](#_Toc21443214)

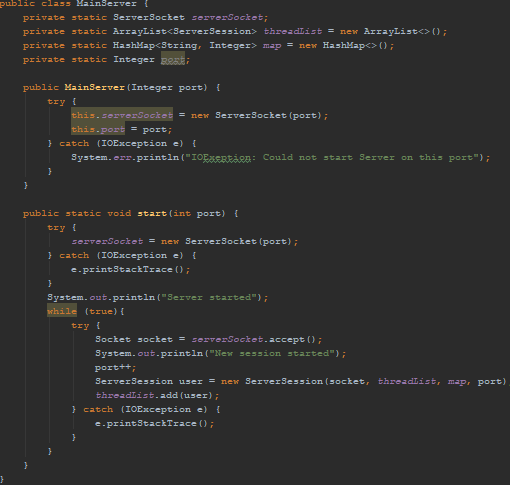
# Grundprinzip

Es gibt einen Hauptserver. Die Clients verbinden sich alle auf diesen. Die Subserver verbinden sich auch auf diesen, ebenso wie ein Client. Ein Client startet einen UDP Client, der für die Kommunikation zwischen den Clients zuständig ist.

# Hauptserver

Die Aufgabe des Hauptservers ist, auf Clients zu warten. Falls sich einer zu ihm verbindet startet dieser eine Session für diesen

## Codeerklärung

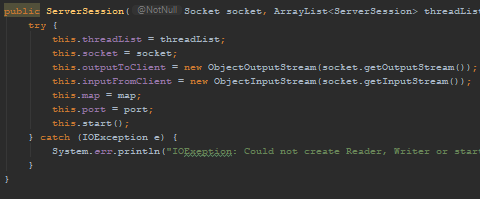


In einer Endlosschleife wird auf Clients gewartet die sich womöglich auf den Server verbinden. Die accept Methode ist eine blockierende Methode. Bei dieser Methode wird erst weiter gegangen, sobald sich ein Client auf den Server verbindet. Wenn sich ein Client auf den Server verbindet, wird der Port des Servers um 1 erhöht. Eine neue Serversession wird gestartet und ihr wird der gerade inkrementierte Port, der Socket, um die Input- und Outputstreams zu erhalten, eine Arraylist, wo sich alle Serversessions befinden und eine Hashmap für später übergeben. Die Hashmap und die Arraylist wird nur übergeben, damit sie unter allen Serversessions synchronisiert ist. Am Ende wird die gestartete Session noch der Arraylist an Sessions beigefügt.

Serversession

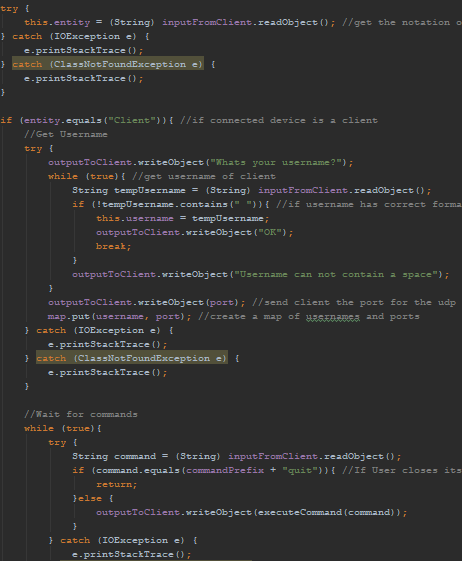
Die Serversession ist die Klasse, die so gut wie alles macht. In dieser werden die Befehle verarbeitet und die Rechenarbeit wird dort gemacht. In dieser Klasse werden die Subserver sowie die Clients verwaltet.

### Konstruktor

Im Konstruktor wird nichts besonderes gemacht, außer, dass ein ObjectInput / ObjectOutput Stream mit dem Socket der übergeben wurde generiert.

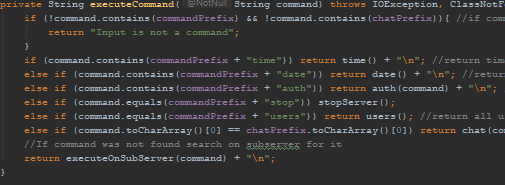
Am Ende wird noch der Thread gestartet.

### Run

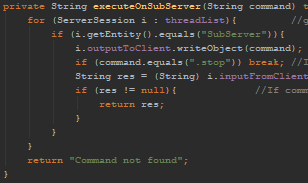


Im ersten Bereich wird ein Verbindungsaufbau zwischen dem Server und dem Client/Subserver vollzogen. Dieses ist besser im Punkt [Verbindungsaufbau](#_Verbindungsaufbau) beschreiben.  
Im zweiten Bereich wird, falls das verbundene Gerät ein Client ist, ein Username von diesem abgefragt. Falls dieser Nutzername ein Leerzeichen enthält wird dem Client eine Fehlermeldung geschickt.  
Im dritten Bereich wird dem Client der Port mitgeteilt, auf welchem er seine UPD Verbindung aufbauen soll. Dieser Port wird dann mit dem Username in eine Hashmap, die jede Serversession hat, eingetragen, wobei der Username hierbei der Schlüssel ist und der Port der Wert.  
Im vierten Bereich wartet die Serversession auf Befehle von einem Client. Jeder Befehl muss das eingestellte Präfix besitzen. Falls der Client das Kommando „quit“ schick beendet sich die Session. Andernfalls wird der Befehl ausgeführt und das Ergebnis zurückgegeben.

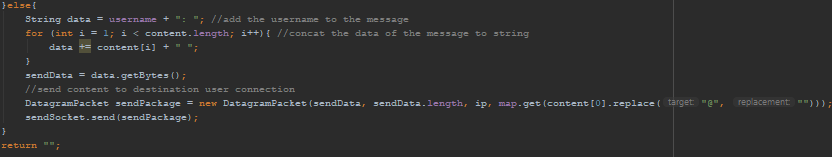
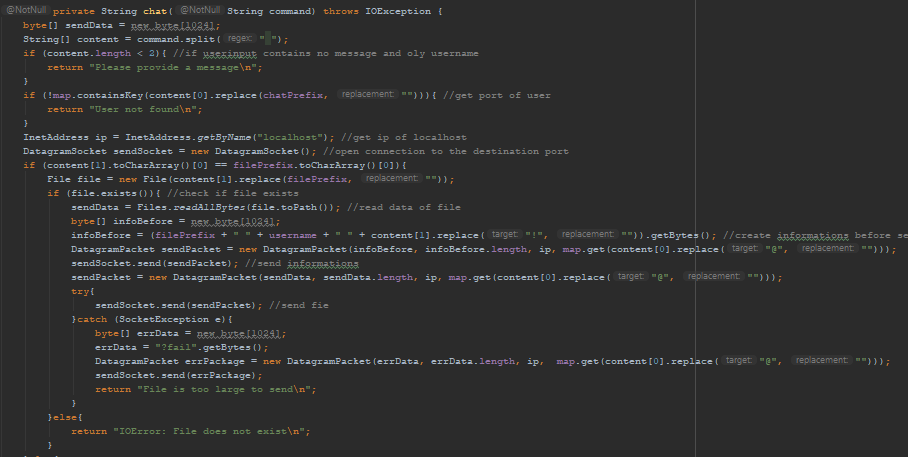
### ExecuteCommand

 Diese Methode wird ausgeführt falls der Client einen Befehl schickt.  
Falls dieser Befehl ohne Befehlspräfix oder Präfix um mit einem anderen Client zu kommunizieren geschickt wurde gibt der Server eine Fehlermeldung zurück (siehe Bereich 1).   
Falls der gesendete Befehl nicht gefunden wurde wird auf den Subservern weitergesucht (siehe Bereich 2).

### ExecuteOnSubserver

Diese Methode geht durch alle Geräte die sich zum Server verbunden haben. Falls einer von denen ein Subserver ist wird ihm der Befehl geschickt. Falls der Befehl „stop“ war, wird die Methode beendet. Andernfalls wird auf die Antwort des Subservers gewartet. Falls dieser den Befehl nicht gefunden hat, wird auf dem nächsten Subserver gesucht. Falls keiner der Subserver den Befehl finden konnte wird dies dem Client mitgeteilt.

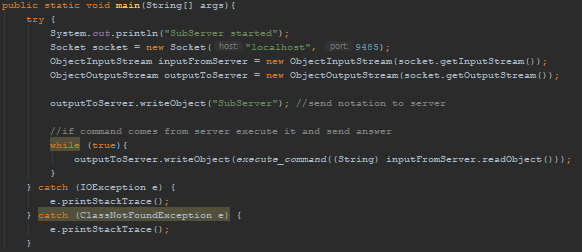
### Chat Methode



Im ersten Bereich wird geschaut ob der eingegebene Befehl des Clients einen Usernamen und eine Nachricht enthält. Falls der Client einen falschen Usernamen oder keine Nachricht angibt wird eine Fehlermeldung geworfen.  
Im zweiten Bereich wird geschaut ob die Nachricht ein Präfix enthält. Falls die Nachricht nicht mit dem Präfix beginnt ist es eine normale Nachricht. Falls dieses Präfix zu Beginn ist wird versucht die Nachricht am angegebenen Pfad zu schicken.  
Falls die Datei existiert werden die Daten der Datei eingelesen. Es werden Zusatzinformationen zusammengehangen und an den Client geschickt. Danach werden noch die Daten der Datei geschickt.  
Falls die Datei nicht existiert wird eine Fehlermeldung geworfen. Falls die Datei zu groß zum schicken war wird das Schicken abgebrochen und eine Fehlermeldung geworfen.  
Im letzten Bereich wird die Nachricht, die der Client eingegeben hat wird hinter den eigenen Usernamen geschrieben und geschickt.

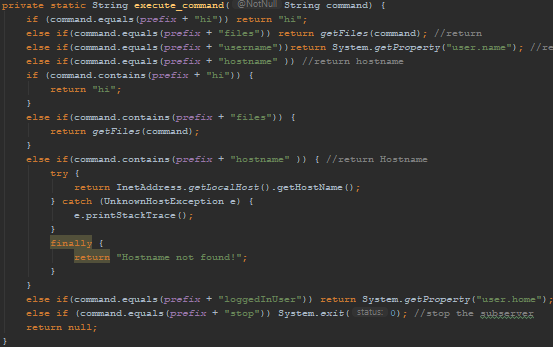
## Subserver / Subserver2

### Main



Starte den Subserver und schicke dem Server den String „Subserver“.  
Warte das der Server einen Befehl schickt. Suche den Befehl und gib das Ergebnis zurück.

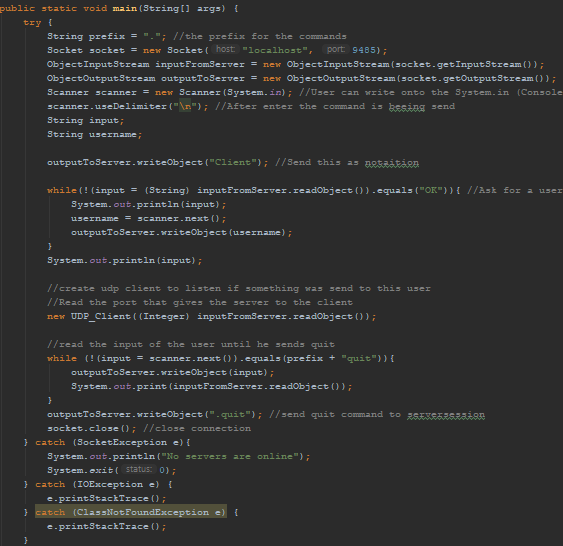
### Execute\_command



Falls der Befehl, der übergeben wurde gefunden wurde gib das Ergebnis zurück. Falls nicht gib „null“ zurück.

## Client

### Main



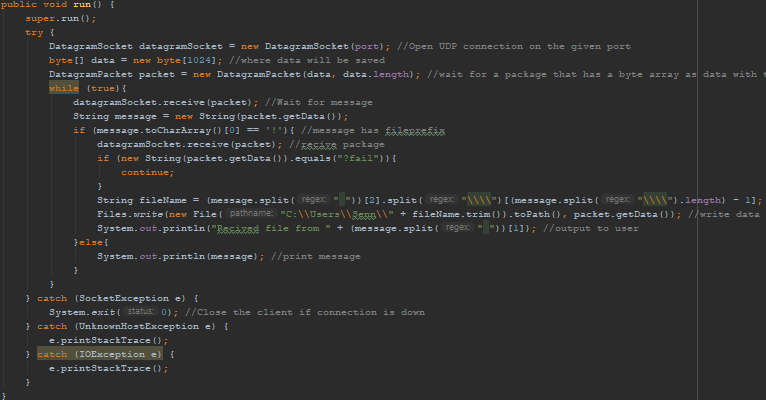
Bereich 1 definiert einen InputStream, einen OutputStream und einen Socket. Es wird „Enter“ als Abbruchbedingung bei für die Eingabe eingestellt. Als Eingabe wird die Konsole benutzt, die sie beim Ausführen öffnet.  
Schicke dem Server „Client“ für den [Verbindungsaufbau](#_Verbindungsaufbau) (Bereich 2).  
Im Bereich 3 muss der Client so lange einen Username eingeben, bis der Server auf den Username ein „OK“ antwortet.  
Im 4. Bereich wird der UDP Client gestartet mit dem Port den der Client vom Server bekommt.  
Der nächste Bereich ist für das Einlesen der Befehle die an den Server geschickt werden. Falls der eingegebene Befehl „quit“ ist wird das Einlesen der Befehle gestoppt.  
Im letzten Bereich (sobald ein Client „quit“ eingegeben hat) wird dem Server ein Quit Befehl geschickt sodass sich die Serversession beendet. Danach wird der Socket des Clients beendet.

## UDP\_Client

### Konstruktor

Im Konstruktor wird der UDP\_Client Thread gestartet.

### Run



Im ersten Bereich wird ein UDP Socket gestartet auf dem Port der übergeben wurde. Erstelle ein Paket, das Empfangen werden kann.  
Im zweiten Bereich wird gewartet, dass ein Client etwas schickt. Falls das geschickte den Filepräfix entfällt wird die gesendete Nachricht ausgewertet du der Dateiname wird herausgelesen. Danach wird auf eine weitere Nachricht gewartet, die die Datei beinhaltet. Falls diese „?fail“ ist wird das Erhalter der Datei beendet. Diese Datei wird dann auf ein fix eingestelltes Verzeichnis gespeichert.  
Der dritte Bereich wird ausgeführt falls eine normale Nachricht gesendet wurde. Dieser Bereich gibt die Nachricht nur aus.

## Zusatzinformationen

### Verbindungsaufbau: Server – andere Geräte (TCP)

1. Ein Gerät verbindet sich auf den Server
2. Das Gerät sendet dem Server was es ist Beispiel: Client oder Subserver
3. Falls das Gerät ein Client ist werden diese Schritte ebenfalls durchgeführt
4. Client gibt einen Usernamen ein
5. Server sendet Client den Port

### Dateien zwischen Clients schicken (UDP)

Um eine Datei per UDP zwischen Clients zu senden muss der Client1 eine Nachricht mit dem folgenden Schema senden: „@username !Pfad\_zur\_Datei“. Falls die Datei existiert und kleiner als 1kb ist kann sie gesendet werden. Wenn ein Client eine Datei erhalten hat bekommt er eine Nachricht, dass er von welchem Nutzer er eine Datei bekommen hat.

### Technische Umsetzung

Als erstes wird überprüft, ob der Nutzer eine Datei verschicken will. Wenn ja wird überprüft ob es die Datei gibt. Wenn es diese gibt wird der Inhalt eingelesen. Eine Zusatzinformation wird erstellt. Diese enthält ein Präfix, den Dateinamen und den Usernamen dessen Clients welcher die Datei verschickt. Die Zusatzinformationen werden direkt geschickt. Danach werden die Daten der Datei geschickt, falls sie nicht zu groß ist. Sobald diese Daten bei Client2 angekommen sind wird dort in einem fixen Verzeichnis eine neue Datei erstellt und die erhaltenen Daten werden eingefügt. Daraufhin wird dem Client2 eine Meldung ausgegeben, dass er vom Client1 eine Datei erhalten hat.

## Persönliche Meinung/Anmerkung

Mir ist das Erstellen dieses Programms bzw. dieser Programme nicht sehr schwergefallen. Ich hatte keine größeren Schwierigkeiten bei dem Teil mit dem Server, den Subservern und den Clients (alles was TCP betraf). Beim Teil wo es um die Kommunikation zwischen den Clients ging hatte ich etwas Probleme beim Aufstellen der Struktur. Bei der Umsetzung (Programmierung) hatte ich keinerlei Probleme. Ich habe dieses Programm bzw. diese Programme in Zusammenarbeit mit Caldato Marcel und Elias Thaler über Github umgesetzt. Bei den Punkten die es als Aufgabe gab habe ich einige kleinere Änderungen vorgenommen, wie die „help“ Methode weggelassen. Ebenso wird der Datagramm Socket direkt gestartet, da auch nicht authentifizierte Nutzer chatten können. Das Chatten habe ich nicht mit einem extra Befehl umgesetzt. Ich habe dafür ein anderes Präfix benutzt. Für die Abgabe würde ich mir eine 9/10 geben, da alles Funktioniert und es mir im Ganzen leichtfiel.