Chat rum

Chat

2042-03-22

Projektmedlemmar:
Lukas Hamrin Brohult <Lukha243@student.liu.se>

Handledare:
Handledare <handledare@liu.se>

Innehåll

Projektplan

1. Introduktion till projektet

Internet Relay Chat (IRC) är ett protokoll för chattmeddelanden. Protokollet använder sig av en klient-server-arkitektur. Det vill säga det finns en server som flera klienter kommunicerar med istället för att kommunicera mellan varandra. Protokollet tillåter att man skriver till kanaler, öppna meddelanden för alla på kanalen, skriver direktmeddelanden till andra användare och mycket mer.

2. Ytterligare bakgrundsinformation

Bakgrunds info finns nedan.
Se Wikipedias artikel: https://en.wikipedia.org/wiki/Internet_Relay_Chat.

Samt IRC protokollet för klienter: https://tools.ietf.org/html/rfc2812

3. Milstolpar

#	Beskrivning
1	**Forska i hur man gör en enkel chattrum!** Titta up i hur man gör en enkel chat rum och sedan få en generel överblick på hur jag ska kunna göra framtida projekt
2	**Ansluta och starta en server!** I detta steg så ska jag få upp en server där man kan gå med i och gå ut ur.
3	**Skapa en client socket** Nästa steg kommer vara att skapa en client socket som kommer prata med min server. Tex ge över meddelanden och andra typer av requests
4	**Skapa olika typer av "request and data objeckt"** Här ska jag skapa några enkla data typer som båda server och client ska använda säg av. Detta för att göra det enkelt för att sicka över data
5	**Ta emot enkla request** På serversidan så ska vi kunna ta emot enkla request och messages och kunna sicka vidare det till resten av våra koplade clients
6	**Skapa en enkel GUI** För att göra det enklare att intragera med så kan vi skappa en enkel meny för medelanden
7	**Gör så gamla medelanden sickas** Om en client joinar senare så bör dem senaste 10-20 medelanden sickas till den nya clienten
8	**Gör så man kan ladda även äldre data** Gör så användaren kan ladda in även äldre medelanden om användaren så vill
9	**Gör så användaren kan gå med i andra kanaler och skapa egna** Gör så användaren kan skapa en egen kanal och också låta andra gå med i sin kanal
10	**Gör så det finns lössen ord på kanalen om man vill det**

	Gör så användaren kan göra ett lössenord på sin egna chat så man måste ha det för att gå med
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
...	

4. Övriga implementationsförberedelser

Planen är att låta användaren gå med i en chat och skriva medelanden.
Jag tror det viktiga är att byga många olika klasser som både servern och clienten kan använda säg av. Jag tror också att bygga projecktet så öppet som går är det viktigaste. Altså att klasserna har funktionerna som går utåt och det är det enda. Med det så kan man enkelt limitera förviring när man bygger andra system som ui och annat

Projektrapport

6. Implementationsbeskrivning

6.1. Milstolpar

6.1.1 Forska i hur man gör ett enkelt chattrum!

Jag under detta steg tog reda på hur man gör ett enkelt chattrum. Jag sökte online på andras lösningar och hur man kan göra det på ett smidigt sätt. Jag hade redan en rätt bra grundkunskap i hur TCP uppkopplingar fungerade och jag kunde enkelt bara absorbera hur man gjorde program i Java. Detta steg hjälpt en hel del

6.1.2 Ansluta och starta en server!

Jag fixade rätt snabbt en uppkoppling mellan klient och server, med detta kunde jag börja skicka enkla saker mellan klient och server och ta reda på hur lätt kontra svårt det skulle bli.

6.1.3 Skapa en client socket

När jag hade fixat en klass för servern så började jag med att göra en klass för klienten där man enkelt kunde skriva in ip:n man ville koppla upp mot servern.

6.1.4 Skapa olika typer av "request and data object"

Mitt request/packet system ändras mycket under tiden av min utveckling. Men nu heter varje "request" ett packet. Dessa packets skickar allt allt från nya meddelanden till om man vill lämna chatten. Det finns en hel hög med packets som skickar instruktioner mellan klienten och servern.

6.1.5 Ta emot enkla request

Detta system tar emot enkla packets och tar hand om dem. Gör olika saker beroende på vilket packet det är. Detta system blev implementerat på klientsidan så både server och klient tar emot data på samma sätt.

6.1.6 Skapa en enkel GUI

Det finns ett GUI som är gjort med Jframe. Den fungerar med vår client klass

6.1.7 Gör så gamla meddelanden skickas

Detta görs med en packet. Det fungerar bra.

6.1.8 Gör så man kan ladda även äldre data

Detta fungerar bra, den skickar samma typ av packet och frågar bara om äldre meddelanden

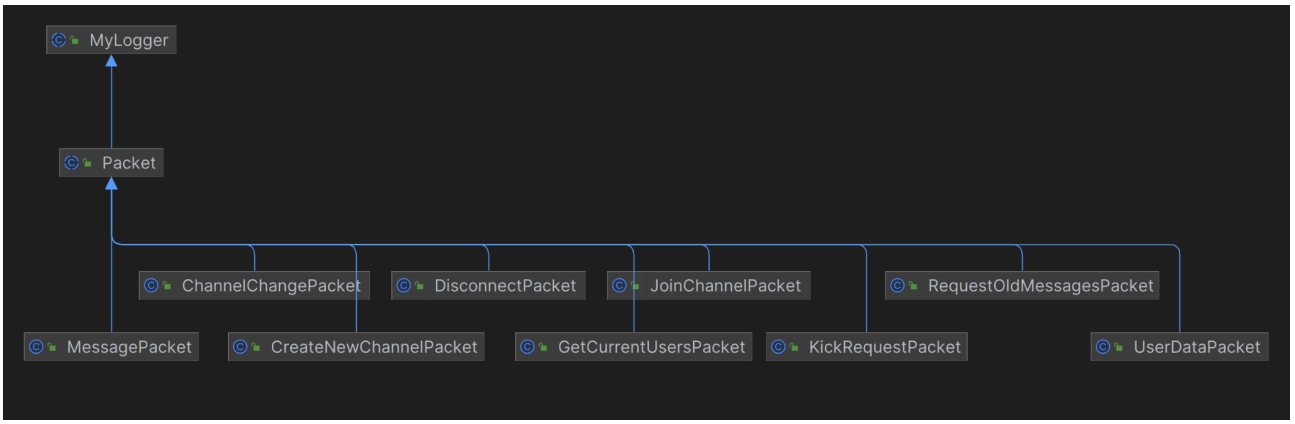
6.1.9 Gör så användaren kan gå med i andra kanaler och skapa egna

Detta görs med packet systemet

6.1.10 Gör så det finns lösenord på kanalen om man vill det

Det går också att sätta ett lösenord på chatten man själv har skapat

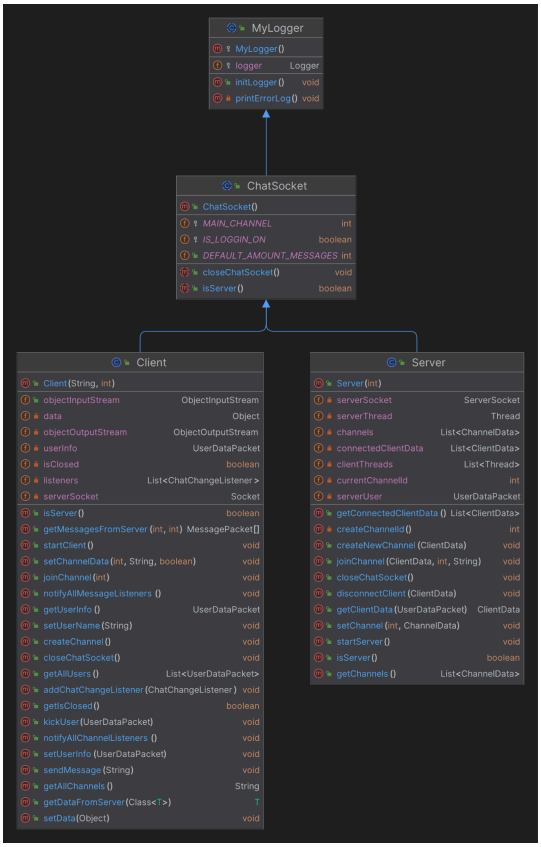
6.2. Dokumentation för programstruktur, med UML-diagram



Packet systemet

Mitt packet system fungerar på det sättet att varje typ av packet extendar den abstrakta klassen *Packet.* På den abstrakta nivån så har vi en funktion som heter *RunServer* och *RunClient.* Dessa metoder blir kallade beroende på om man vill komma åt serverversionen eller client-versionen då vi vill att olika saker ska hända beroende på vilken som blir kallad. Den abstrakta nivån extendar också med *MyLogger* som många klasser gör. Detta är för att dem ska ha tillgång till loggermetoden. Den implementerar också *Serializable-klassen* som är en inbyggd funktion i Javas “Java IO" bibliotek. Denna klass gör att vi enkelt sen kan skicka alla dessa klasser över internet utan att serialisera dem själva och det underlättar massa.

Alla våra paket innehåller både våra två metoder men också specifika fält där vi sparar datan vi vill skicka över vår ström till den andra socketen. Som till exempel i *MessagePacket* så har vi fältet “string message” där den sparar meddelandet av användaren i detta fall. Det skickas sen över nätet och med att vår huvudklass (Packet) implementerar serializable så löser den själv att göra om våran string till bytes.



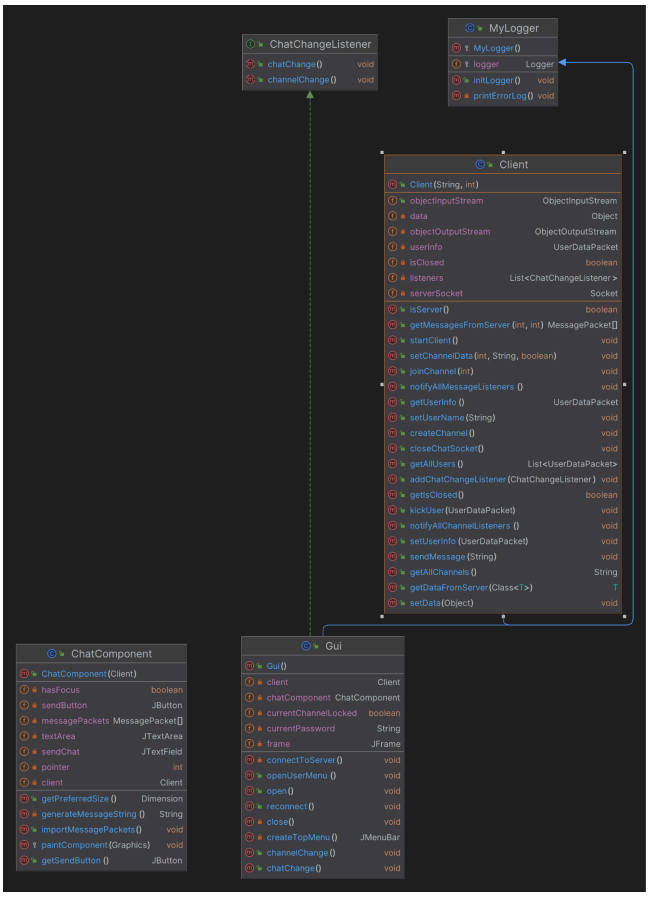
Chat socket systemet

I vårt chat socket system har vi en huvudklass som är abstrakt, denna extender *MyLogger* samt så har den olika metoder som passar in på båda typerna av sockets. Vi har två typer av sockets, vår *Server* klass och *Client* klass. Vår *Server-klass* är vår server för vår chatt, och den tar emot uppkopplingar från vår Client-klass. I båda klasserna så finns det en socket som är en färdigbyggd klass från packet "java io". Den tar hand om att skapa en koppling mellan olika klienter.

I vår server så har vi en privat klass som heter *Connector* som kör en runnable på en egen tråd. I den så tittar den efter nya användare som går med i chat-systemet. När någon går med så kommer servern skapa en egen tråd för den användaren samt skapa en *ClientData* där den sparar server specifik data. Som dess output ström och input ström. Med denna data så skapar vi en ny tråd med klassen *Receiver* som från nu tar hand om att ta emot packets från just den användaren.

På Client-klassen funkar det rätt likt förutom att vi inte har en *Connector* då det inte krävs, vi behöver bara koppla upp oss en gång. Så den har också en *Receiver* tråd som tar emot packet från servern. Tex när en annan användare så skickar servern vidare det till klienterna med.

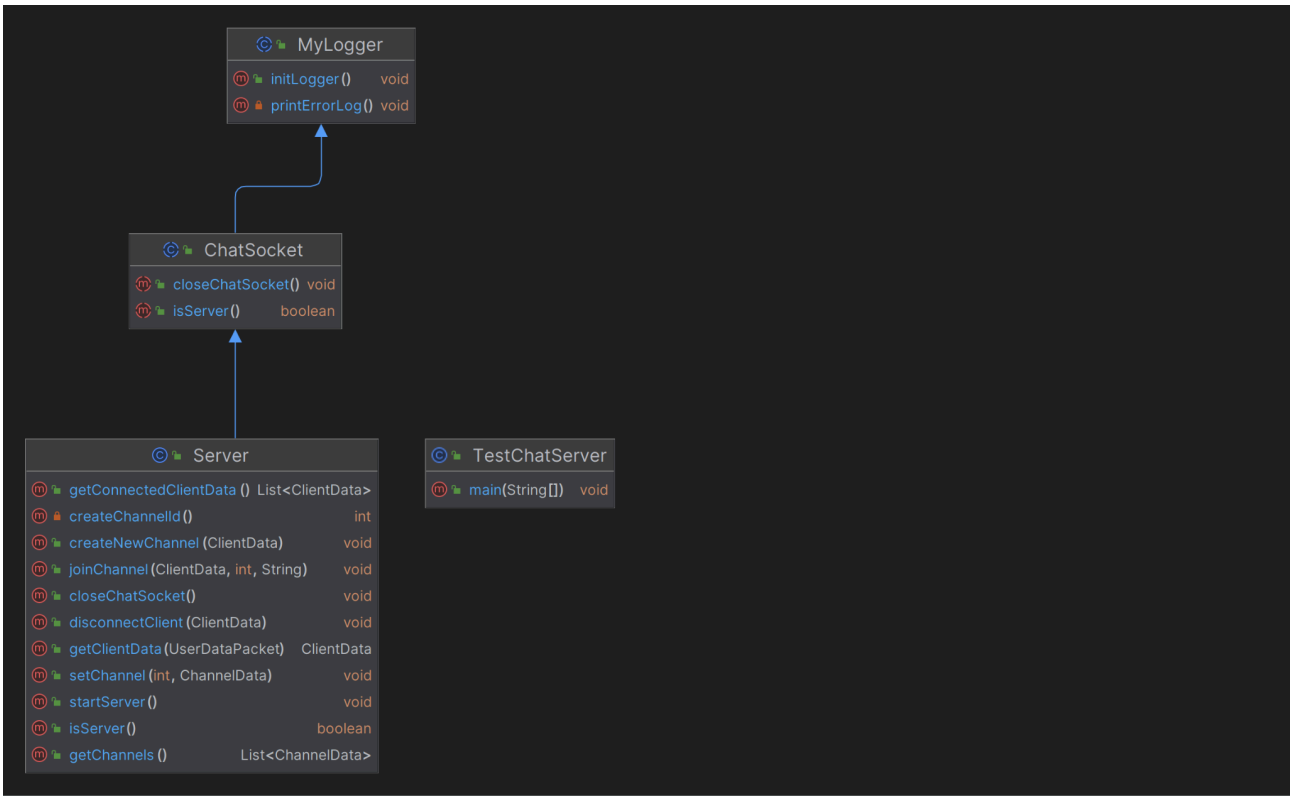
I Client-klassen har vi också ett annat sätt att få tag på data:an man får igenom ett paket. För att man bara ska ha en input ström så kan vi inte använda oss av "readObject()" mer en gång. Annars började den strula. Det löste jag genom att sätta datan till ett objekt i client-klassen. Så i varje funktion där jag inväntar data från servern så väntar den på att få in datan från servern på detta sätt. Detta är gjort i "getDataFromServer()" som inväntar att datan du väntar på sätts. Jag tycker inte att detta är en mycket bra lösning men det är vad jag hade tid med att göra! Det fungerar också rätt bra.



Client side

För client side så har vi ett *GUI* så att man kan använda klientens olika metoder. I vårt *GUI* så skapar jag en *Client* klass som vi sedan kör connect metoden för att koppla upp sig mot utvald ip..Sedan använder man sig av dem olika metoderna i *Client* för att kunna göra olika grejer i vårt GUI. Till exempel som att skicka ett meddelande kallar “sendMessage” i *Client* som sedan tar hand om logiken bakom att skicka vårt meddelande.

Vi har också en *ChatChangeListener* interface som har två metoder. En när man har fått ett nytt meddelande och ett när hela chatten uppdateras. Dessa kallas från olika utvalda paket. Till exempel när man får in ett nytt meddelande så vet GUI:it att det behöver uppdatera och sätta in de nya meddelandet



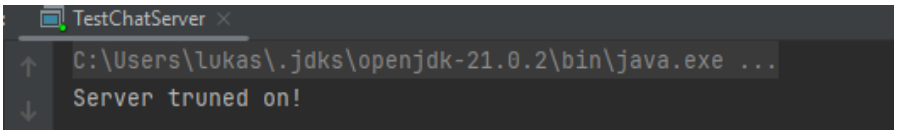
Server programet

En av de viktigaste men också en av de lättast strukturerna. Vi startar programmet och den startar igång vår server. Från den punkten inväntar den nya klienten att gå med.

7. Användarmanual

Sätta igång programmet:
Steg 1 .

Starta filen *TestChatServer* och invänta att det står “Server turned on!” i consolen.



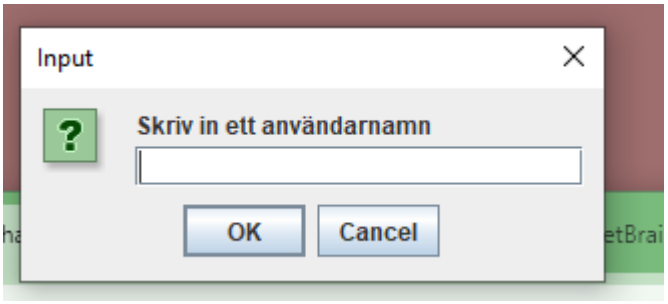
Steg 2 .

Sedan kan du starta TestChatClient1 och då ska du få upp en ruta där den ber dig om en IP

Du kan lämna denna ruta tom eller skriva in ens lokala ip 127.0.0.1. För att då testa lokalt

Steg 3.

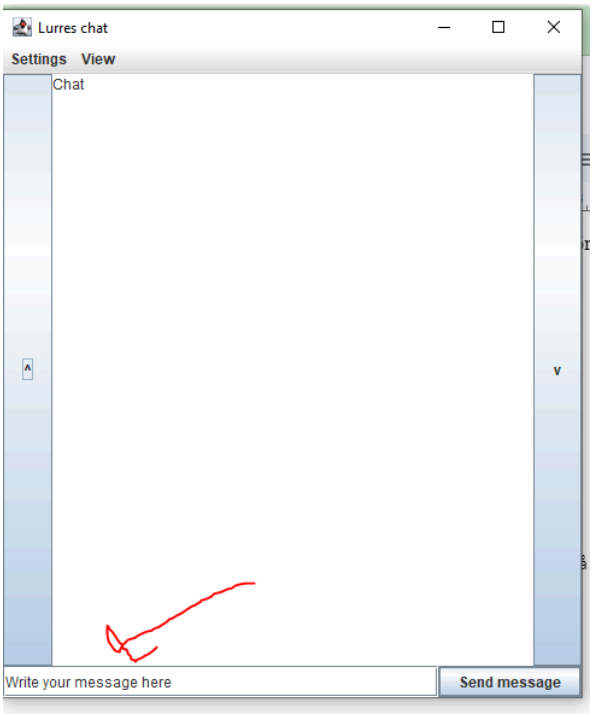
Sedan kan du skriva in ett användarnamn



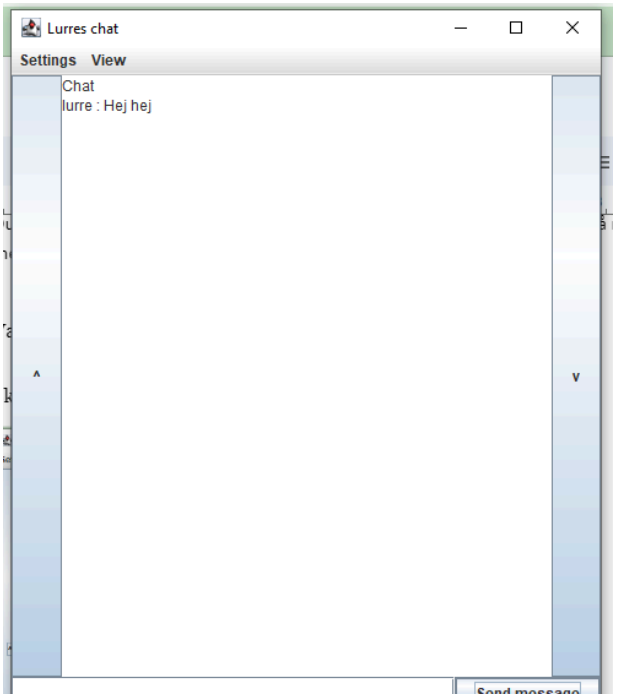
Du är nu inne i chat servicen. Om du vill ha en andra klient för att kunna se så man kan skicka meddelanden mellan dem två så kan du köra *TestChatClient2*

Vad kan man göra när man har kommit in?

Skicka medelanden:

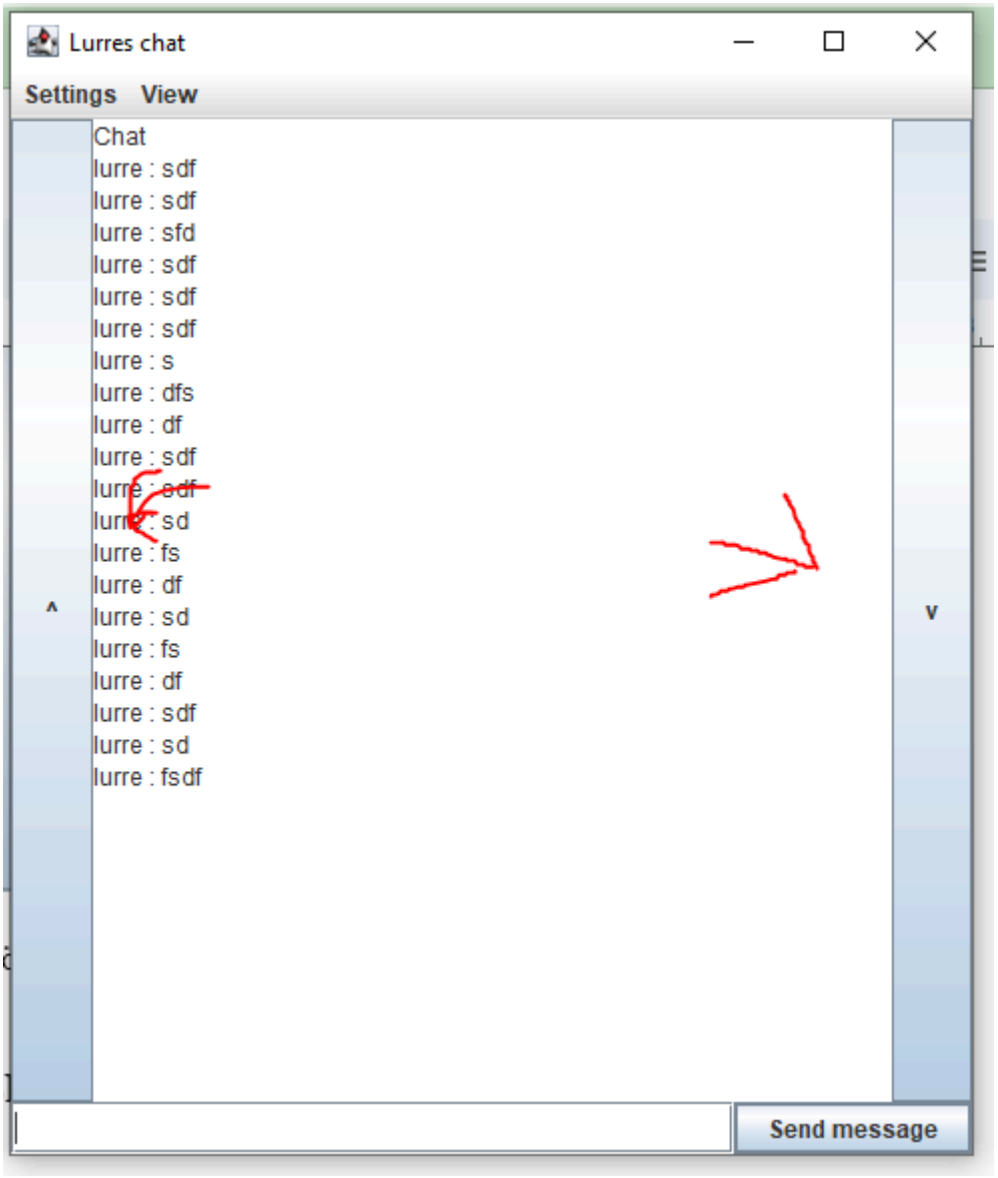


Skriv in meddelanden här sedan klicka på knappen bredvid för att skicka iväg



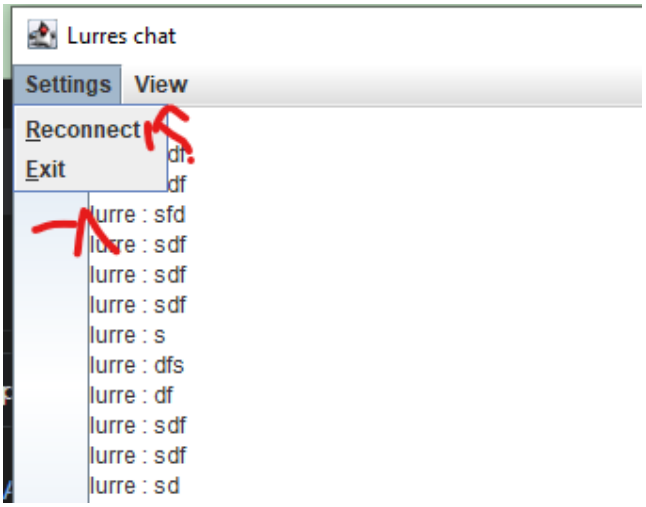
Bör se ut så här efter

Bläddra äldre meddelanden

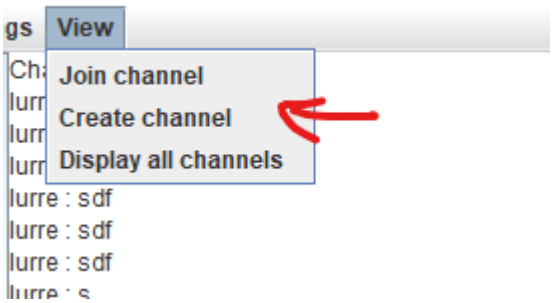


För att se äldre så klickar man på vänster. Då går man upp med 5 meddelanden per klick. Den vänstra tar dig ner till mera nuvarande och nya meddelanden.

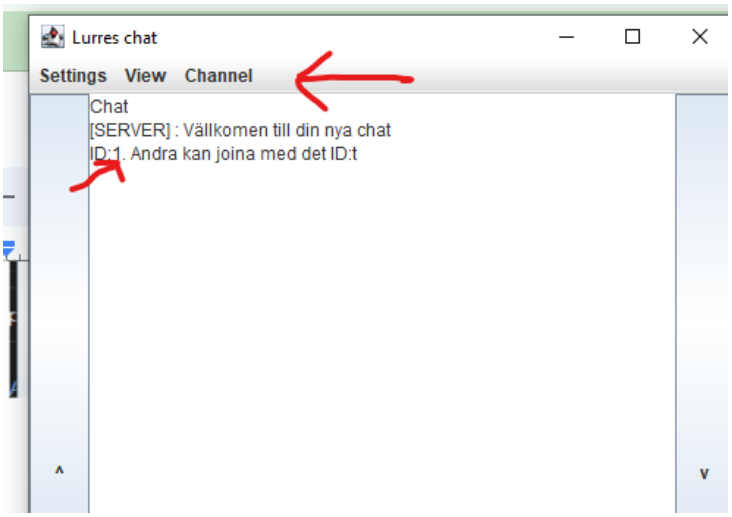
För att lämna och gå med i en annan ip igen:



Skappa en egen chat

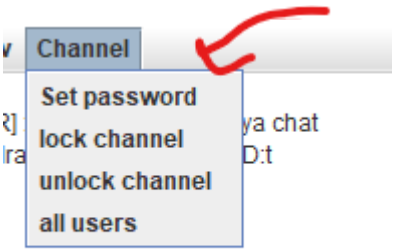


Klicka på create channel.

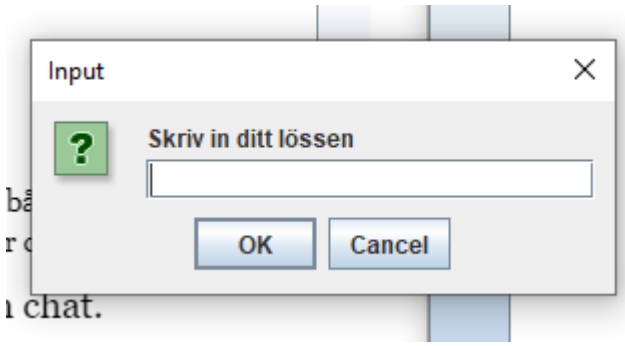


Du bör få en chat där du får både id och sen får du en kanal som chat ägare som har några olika inställningar du kan sätta för chatten.

Sätta lösenord på egen chat.

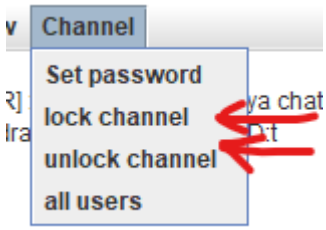


Klicka på “Set password”



Skriv in sedan valt lössen. Din kanal kommer fortfarande vara öppen. Du behöver låsa den för att det ska krävas ett lössen.

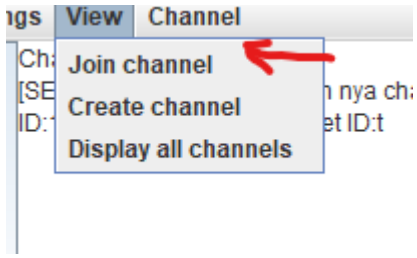
Låsa och låsa upp



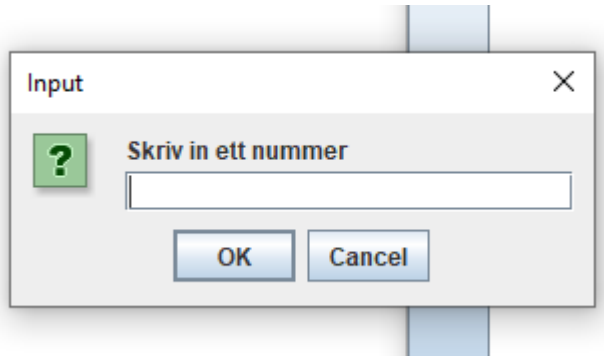
Klicka på “lock channel” för att låsa

Klicka på “unlock channel” för att låsa upp

Gå med i en annans person chat



Tryck på “join channel”



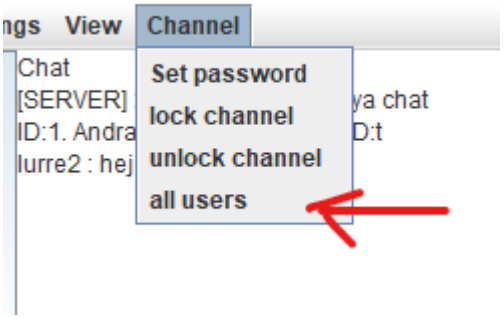
Skriv sedan in ID för kanalen du vill gå med i. 0 är alltid första kanalen

Se alla kannaler

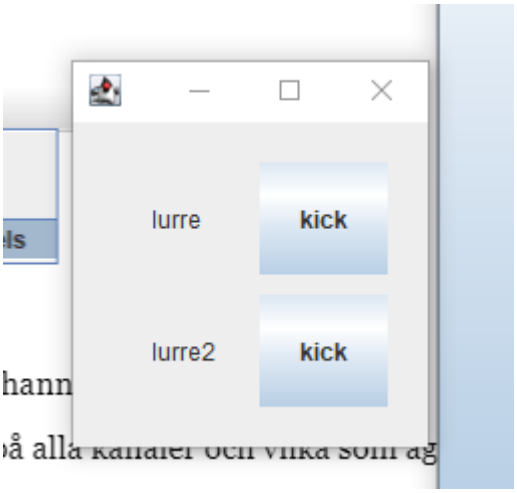
Klicka på “Display all channels”

Då får du upp en lista på alla kanaler och vilka som äger respektive och deras ID.

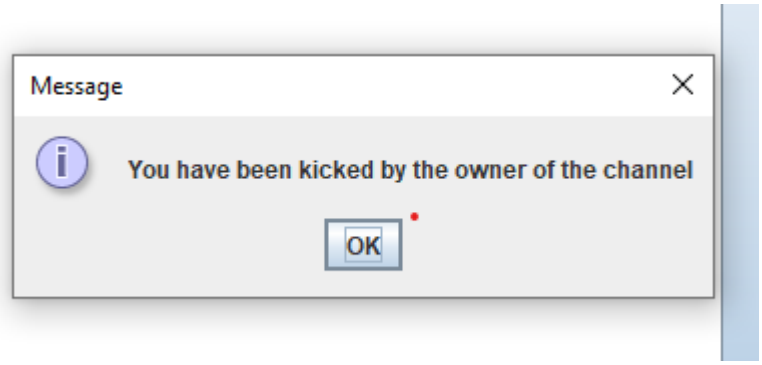
Kicka user från din kanal och se alla som är i den kannal



Klicka på “all users”



Du får upp alla som är i din chatt och du kan trycka på kick för att kasta ut någon ur din chat!



Det är alla funktioner. Tack för att du har läst :D