Programdesign  
  
Vi skapade två separata program. En klient och en server. På serversidan hade vi fyra klasser: Server, ClientConnection, Subject och Observer. Server implementerar Observer interfacet och ClientConnection är en subklass till Subject klassen. Servern initialiseras för att sedan sitta och vänta på klienter som kopplar upp sig till den. När en klient väl kopplat upp sig mot servern startas en ny instans av ClientConnection och servern registreras som observer för den instansen. Då varje server anslutning kommer att sitta och vänta på user input så valde vi att implementera Runnable interfacet och skapa en egen tråd som startas av Servern i samband med att en ny anslutning upprättats.  
  
När en uppkopplad användare skriver något så anropas update metoden som Servern deklarerat vilket i sin tur itererar genom en lista av alla nuvarande anslutningar. När den stöter på det nya meddelandet så skickar den det till alla klienter. Därefter så nollställs meddelandet så att det inte skickas igen vid nästa anrop av serverns update metod.  
  
På klientsidan så har vi sex klasser: Client, GUI, PropHandling, Observer, Subject och en StartClass. Client klassen har i uppgift att stå för kopplingen mellan klienten och servern. StartClass  
initialiserar ett nytt objekt av Client klassen och skickar med det som referens när den initialiserar GUI klassen. Client är en subklass till Subject och GUI implementerar Observer. I GUIns konstruktor sätter GUIt sig själv som observer för den instans av Client som skickades med som referens. Om användaren trycker på Send knappen eller trycker på Enter på tangenbordet så skickar GUIt ett kall till Cliet med innehållet i det textfält man skriver i.

Då klienten hela tiden måste sitta och vänta på input från Servern så valde vi att starta en ny tråd för den metoden och ange en anonym Runnable instans som argument. I den så finns det en oändlig loop som väntar på input från servern. När en ny input kommer från InStream (Servern) så anropas notifyObserver() vilket i det här fallet gör så att meddelandet läggs till på och uppdaterar den TextArea som visas på skärmen. Anslutning tills servern sker genom att StartClass anropar connect metoden för Client. Som argument anges IP adress och port vilket hämtas från en Java Property file.

Problem

I ett kort tag så had vi problem på Server-sidan med att två trådar ändrade samma variabel samtidigt, vilket vi senare löste genom att implementera Observer-mönstret.

Två utav medlemmarna hade problem att få Github att fungera, men när det väl var igång så flöt det på.

<https://github.com/Beardstorm/ChatService.git>