Chat client-server - documentazione

Chat client-server 1.0

Passaggi per la connessione:
Creazione del server
Creazione del client che si dovrà collegare al server
Dopo aver instaurato la connessione il client invia il nome al server
Il client invia il messaggio al server, questo lo invia all'altro client
Viceversa il secondo client deve inviarlo al primo
per uscire dalla chat scrivere "Logout"

Classe Server
classe Client
classe per la gestione dei client

La prima classe viene usata per creare il server, ricevere e accettare le varie richieste di comunicazione da parte dei client

La seconda classe verrà usata per inserire i nomi dei client all'interno del vettore per usarli in seguito nella ricerca del destinatario.
Qui utilizzo Stringtokenizer per separare il messaggio dal destinatario tramite il formato:
messaggio#destinatario

Classe Client
La classe contiene 2 thread, uno per l'invio del messaggio, l'altro per la lettura del messaggio.
WriteUTF e ReadUTF utilizzano la codifica UTF-8 ovvero una codifica usata per rappresentare i caratteri Unicode come **#** o **:**
Il **#** viene usato come separatore tra il messaggio e il destinatario.

Chat client-server 2.0

Primo obiettivo: gestione della disconnessione del client

```
if(ricevuta.equals("Logout")){
                this.login=false;
                this.s.close();
                break;
            }
```

Il comando Logout, quando viene digitato dal client, blocca la chat e invia un messaggio al server in cui viene che il client è uscito dalla chat e non è più raggiungibile.

Conclusioni sull'obiettivo
Tutti i messaggi inviati dal client si bloccano e non vengono mostrati neanche sull'output del server, ciò include anche anche il Logout.

Secondo obiettivo: i client devono conoscere i nomi dei client connessi

Per fare ciò invio al client connesso ogni client mediante l'uso della variabile i:

IDEA INIZIALE

```
if(i == 0){
            System.out.println("Sei il Client 0 e sei il primo nella chat");
          }
          else if(i > 0){
            System.out.println("Sei il Client " + i);
            while(i != 0){
              System.out.println("Nella chat è presente: Client " + i - -);
            }
          }
```

```
{
… //varie righe di codice
}
```

```
i++;
```

Conclusioni dell'idea iniziale: facendo così c'è il problema che, una volta che i diminuisce, resti a 0 e quando aumenta torna a 1 rimanendo in un ciclo infinito

---

```
if(i == 0){
            System.out.println("Sei il Client 0 e sei il primo nella chat");
          }
          else if(x > 0){
            System.out.println("Sei il Client " + i);
int x = i;
            while(x != 0){
               System.out.println("Nella chat è presente: Client " + x - -);
            }
```

```
        }
{
… //varie righe di codice
}

i++;
```

Aggiungo una variabile x = i in modo tale da usare x solo per diminuirla e mostrare ai client tutti quelli già connessi nella chat, mentre così i rimane con lo stesso valore senza modificarla. Quando x viene diminuita, prima del while, viene posta nuovamente uguale a i.

Per far mostrare al client i vari messaggi del server riguardante chi è connesso nella chat, sostituisco i vari System.out.println precedenti con dis.writeBytes() che mi consente di inviare la stringa del messaggio al client. Nella classe client ho creato un metodo per poter mostrare all'utente il messaggio con un System.out.println().

Conclusioni sull'obiettivo

Il programma funziona ma la parte di codice dove si trova il while viene completamente saltata dal programma, mentre a volte lo esegue ma saltando parti del messaggio, ad esempio: at è presente: Client 1.
L'altro problema è che non vengono mostrati i precedenti Client nella chat, come dovrebbe fare dato che diminuisco sempre la x, ma anzi mostra il numero dello stesso Client.

Terzo obiettivo: possibilità di avviare chat one to one oppure one to all

Il codice sottostante è utilizzato sia nella prima che nella seconda chat client server e viene utilizzata per inviare il messaggio, contenuto nella variabile ricevuta, al client destinatario posto dopo #.

```
StringTokenizer st = new StringTokenizer(ricevuta, "#");
String MsgToSend = st.nextToken();
String recipient = st.nextToken();
```

Nella variabile recipient viene inserito il destinatario.

L'idea per modificare il codice in modo tale da poter inviare il messaggio a tutti è (quella scritta in grassetto) di aggiungere un ulteriore controllo quando il server scorre il vettore, e quando non trova il destinatario il messaggio viene inoltrato a tutti.

```
for (ClientHandler mc : Server.lc){
            if (mc.name.equals(recipient) && mc.login==true){
               mc.dos.writeUTF(this.name+ " : " +MsgToSend);
               break;
            } else if(mc.login==false){
               System.out.println("Il client destinatario è uscito dalla
chat");
            } else if(Server.i <= 1){
               System.out.println("Sei l'unico partecipante alla chat");
            } else if(!mc.name.equals(recipient)){
               mc.dos.writeUTF(MsgToSend);
            }
         }
```

Conclusioni sull'obiettivo

Il messaggio one-to-one inviato da un client a un altro non arriva. Prima di altre modifiche il messaggio non solo non arrivava ma dava anche un errore nell'output del server. Una volta risolto ciò il messaggio è possibile inviarlo con risultato, però, che quest'ultimo non arriva né all'altro client né nell'output del server.
Inoltre il programma Netbeans non segnala alcun errore nel codice e in più sono stati inseriti gli appositi '\n' per andare a capo a ogni riga in cui si trova il readLine e il writeBytes.

Infine, l’invio dei messaggi nella versione precedente della chat client-server funzionava correttamente seguendo anche i vari controlli se il destinatario era ancora presente nella chat o no.