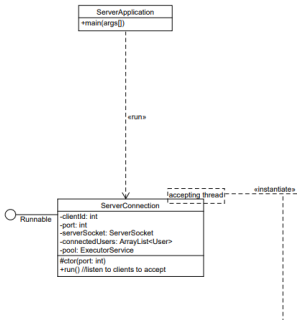
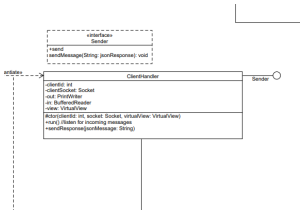
Server Explanation

1 ServerApplication



ServerApplication e' il main, al momento inizializza e fa runnare ServerConnection, qui si fara' uso delle classi di settings. ServerConnection una volta inizializzato con le porte (TCP ed RMI) ed il numero massimo di thread (Client) nel metodo run() sta in ascolto del socket, quando un client viene rilevato inizializza un ClientHandler, una VirtualView ed aggiunge il ClientHandler al ServerController, se tutto e' andato a buon fine fa partire il thread del client.

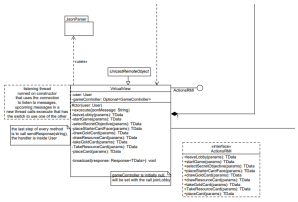
2 ClientHandler



ClientHandler quando viene inizializzato ha la stream di input e di output della socket ed un clientId dato da ServerConnection, questo id e' univoco nel Server. Nel run il thread ascolta la socket, quando un messaggio arriva viene creato un nuovo thread per elaborarlo, e la socket si rimette in ascolto. Il thread in questione chiama il metodo execute della sua virtualView e col valore di ritorno di questo metodo, usa il suo metodo sendResponse per mandare la risposta al client attraverso la socket.

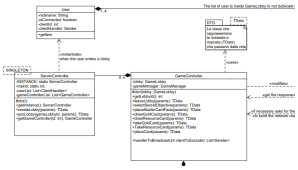
ServerConnection e ClientHandler sono predisposti all'RMI, ma non ancora implementato

3 VirtualView



VirtualView, il metodo excecute prende un json ed e' il dispatcher, a seconda del metod interno alla richiesta verra' chiamata l'implementazione corretta dei metodi interni alla virtual view (Quelli dell'interfaccia ActionsRMI, gli stessi metodi che verranno esposti con l'RMI). Nelle implementazioni dei metodi ci sara' la prima parta che e' una chiamata al controller, la risposta di questa chiamata verra' mandata a tutti i client a cui deve essere mandata tramite il metodo broadcast interno alla virtualView (la lista di clientHandler a cui fare 'sendResponse' viene presa dal controller). Infine il metodo ritornera' il valore nel thread del clientHandler che lo ha invocato, in modo tale da poter mandare la risposta al chiamante del server. In questo modo e' anche possibile facilmente personalizzare le risposte. VirtualView ha un riferimento a gameController nullable per eliminare il collo di bottiglia: ServerController e' un Singleton, poi i client si interfacceranno direttamente con il loro GameController.

4 Controller



ServerController e' un singleton che ocntiene i riferimenti a tutti i clientHandler connessi e gli permette di loggarsi o creare un Game; in quel momento verra' creato un riferimento di tipo User, che serve a mappare i dati di gioco del player (username) alla sua classe per la gestione della rete (ClientHandler, con interfaccia Sender), e creato un nuovo gameController (se createLobby) al quale poi verra' aggiunto lo user. Quando uno di questi metodi ritorna alla VirtualView senza errori viene settato anche il corretto gameController all'interno della stasse.

GameController gestisce il gioco inteso come lobby+partita effettiva. L'unico metodo particolare e' quello che ti permette di ottenere la lista di Sender connessi alla partita (che sono i client al quale la virtual view deve notificare le risposte nel broadcast)