Corso di Reti dei calcolatori
Relazione del progetto “Hangman”

Simone Salvo
matricola 450900

18 agosto 2015

Introduzione

Il progetto

Il progetto richiedeva lo sviluppo del gioco dell'impiccato. Differentemente dalla versione classica del gioco, questo sviluppo doveva permettere la possibilità di gioco multi-utente, cioè più utenti devono poter partecipare ad una partita. Le specifiche dell'applicazione hanno portato allo sviluppo di due processi, una server ed una client. Il processo server gestisce le varie richieste di registrazione/login utente oltre che alle richieste di creazione o di aggregazione di/ad una partita. Il processo client permette all'utente di prendere parte di una partita. Il client si distingue in master o guesser: un client-master fa da gestore della partita, un client-guesser fa giocatore. La differenza a livello di sviluppo è minima, si tratta della stessa applicazione che a seconda che si tratti di un utente che ha creato una partita o meno assume dei comportamenti leggermenti diversi. Una volta avviato il server resta in ascolto di possibili connessioni da parte di un client e gestisce l'apertura o meno di una partita. Una partita può essere aperta o meno se il numero delle partite aperte non supera il limite impostato nel file di configurazione json del server e se l'indirizzo multicast che un client-master invia al server per l'apertura di una partita non è già esistente; infatti pur usando la crittografia per lo scambio dei messaggi che eviterebbe possibili interferenze di messaggi di partite diverse sullo stesso indirizzo broadcast, si è preferito evitare del tutto la situazione in cui più partite possano utilizzare lo stesso indirizzo broadcast. Ad un client-master che proverà ad aprire una partita con uno stesso indirizzo di broadcast verrà negata l'apertura con uno specifico messaggio da parte del server, il client in questione dovrà quindi modificare il suo file di configurazione e riprovare ad aprire la partita. Si è scelto di portare il client sempre alla situazione iniziale di processo (dove per situazione iniziale di processo si intende quella che segue l'apertura dell'applicazione) successivamente ad ogni errore o termine di una partita per rispettare i vincoli progettuali, in particolare lo sviluppatore avrebbe preferito non chiudere le connessioni server-client fino a terminazione dei relativi processi, questo avrebbe permesso una più agevole gestione del gioco permettendo all'utente di uscire e creare una partita senza passare dalla fase di autenticazione ogni volta.

Lo sviluppo

Il progetto è stato sviluppato su un Macbook con OS X 10.9.5 e Java Platform 7. L'IDE utilizzato è stato IntelliJ 14. Librerie esterne di cui si è fatto uso sono state:

- Gson
- Jasypt

- Json-jena
- Json-simple
- Lombok

Per tutte le librerie esterne utilizzate si rimanda ai riferimenti al termine della relazione.

Al fine di poter effettuare la fase di debug e test più agilmente, è stato scelta un'implementazione a moduli, quest'ultimi contenuti all'interno del progetto HANGMAN sono incapsulati dentro un package con il prefisso it.simonesalvo.hangman e con un postfisso che prende il nome dal relativo modulo, i.e. ".client", ".server", etc.

Sia il server che ogni client necessitano della specifica iniziale di un file di configurazione. Questo, oltre che ad essere un file json, deve contenere necessariamente i seguenti attributi con i relativi valori:

- ServerSocketAddress: Indirizzo della socket del server
- ClientAddress: Indirizzo del client, lo stesso utilizzato dalla socket del client
- MulticastAddress: Indirizzo multicast, utilizzato in caso di apertura di una partita
- ClientSocketPort: Numero di porta della socket del processo client
- ServerSocketPort: Numero di porta della socket del processo server
- MulticastPort: Numero di porta multicast
- RMIClientPort: Numero di porta RMI relativamente al client
- RMIServerPort: Numero di porta RMI relativamente al server

Tutti i valori degli attributi di cui sopra sono e devono essere avvalorati come stringhe, al fine di garantire il corretto funzionamento. Gli attributi sopracitati costituiscono l'unione di quelli necessari al server e al clienti per il corretto funzionamento. Quanto appena detto sta a significare che ci sono attributi che possono non essere validi a seconda che si tratti dell'applicazione server o client. Maggiori dettagli sono specificati nelle relative sezioni dei processi client e server.

Oltre ai relativi file di configurazione di server e client un importante file è quello utenti usr.json. Quest'ultimo conserva tutti i nome utente e le password (in formato md5) di tutti gli utenti che hanno creato un account. Questo permette ad un utente registrato in precedenza di effettuare l'accesso successivamente senza registrarsi ma semplicemente re-inserendo nome utente e password utilizzati in fase di registrazione.

Qualsiasi input dell'utente e/o lettura da file viene realizzata usando una codifica UTF-8 al fine di evitare comportamenti anomali con caratteri speciali o simili.

Server

Il processo server nel progetto hangman si occupa di:

- gestione registrazione e autenticazione dei client;
- gestione apertura e notifica di nuove partite.

Le due funzionalità fondamentali del server sono meglio specificate nelle relative sezioni sottostanti. Successivamente alle due sezioni iniziali seguono altre due sezioni: l'adviser e il connection handler. Si tratta di due "meccanismi" di fondamentale importanza e che verranno trattati nei dettagli nelle relative sezioni. L'ultima sezione riguardante il processo server specifica alcuni dettagli che riguardano la chiusura del processo stesso.

Al fine di essere eseguito con successo il server necessità di un file di configurazione in formato json, un esempio di tale file di configurazione è il seguente:

Tutti i campi presenti nell'esempio devono necessariamente essere presenti e correttamente avvalorati.

Registrazione, login, logout dei giocatori

Come da vincolo implementativo, la registrazione, il login e il logout di un giocatore avviene tramite RMI. L'oggetto RMI è "esposto" ai client che quindi può invocare i metodi definiti al suo interno. In particolare i metodi esposti dall'oggetto RMI server sono:

{
"serverSocketAddress":"127.0.0.1",
"serverSocketPort":"9098",
"personalAddress":"127.0.0.1",
"maxGameNumber":"10",
"RMIServerPort":"35009"
}

- Create account: La creazione di un account consiste nella validazione del nome utente e della password che devono essere di minimo 4 e 5 caratteri rispettivamente, successivamente viene verificato che il nome utente non sia già esistente, se tutti i controlli vengono superati correttamente, il nuovo utente viene aggiunto al file utenti. In questo modo l'utente potrà successivamente effettuare il login per effettuare ulteriori accessi al gioco senza la necessità di effettuare una registrazione per ogni accesso.
- Login: L'autenticazione di un account consiste nella validazione del nome utente e della password che in questo caso devono essere contenuti nel file utenti. Se il il login viene effettuato con successo, cioè se il nome utente e la password vengono validati allora l'utente viene considerato come connesso, quindi viene creato un oggetto risultato della connessione tra il server e la callback del relativo client. La callback non è altro che un medesimo della tecnologia usata dal server per esporre i metodi per la registrazione, la login e il logout dei giocatori. In questo caso, l'oggetto RMI in questione serve ad accedere all'unico metodo esposto dal client per l'invio (lato server) e quindi la ricezione (lato client) della lista aggiornata delle partite aperte.
- Logout: Il logout di un account consiste nella rimozione del relativo account tra quelli "connessi" oltre che alla rimozione di ogni riferimento all'oggetto esposto dal client per la comunicazione delle partite disponibili.

Apertura e chiusura di una partita

L'adviser

L'adviser permette di "avvisare" i client con la lista delle partite disponibili. Le partite disponibili non vengono inoltrate a tutti i client, bensì ai soli client non "impegnati". Un client viene definito "impegnato" quando ha già assunto la "qualità" di Guesser o Master[1]. Non tutte le partite sono elette a partite disponibili e quindi inviate agli utenti non "impegnati". Le partite disponibili sono quelle in stato di "HIDLE", cioè partite che non sono iniziate e che sono quindi in attesa del raggiungimento del numero necessario di giocatori. Oltre alla funzione "avvisatrice", l'adviser implementa un task (thread[2]). Il task adviser è cadenzato, ogni 5 secondi dalla sua apertura viene effettua le seguenti azioni:

1. Per ogni partita in stato di HIDLE verifica che questa abbia raggiunto il numero necessario di giocatori;
2. Nel caso in cui una partita abbia raggiunto il numero necessario di giocatori, a tutti, master incluso viene inviato un messaggio che ha come significato quello di indicare che la partita può cominciare. A questo punto la socket che all'atto dell'autenticazione era stata aperta con il client viene chiusa e la partita viene eliminata dal server, il controllo della partita passera al client-master che ne gestirà lo stato. Una volta che una partita è iniziata tutti i client coinvolti vengono "sloggati" dal server, in questo modo si garantisce che nel caso in cui uno di

[1] Guesser e Master sono definiti della specifica del progetto Hangman

[2] A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.

questi client voglia continuare a giocare aprendo un'altra partita o aggregandosi a una partita precedentemente aperta, questo dovrà effettuare nuovamente il login o passare attraverso la creazione di un account.

3. Per ogni partita in stato di ANNULLED viene propagato un messaggio a tutti i client relativi a quella partita per informarli che la partita è stata annullata. L'annullamento di una partita avviene per l'uscita anticipata da parte del client-master, a partita non ancora iniziata. Una volta progata l'informazione di annullamento della partita, la partita viene eliminata dal server e i relativi utenti "sloggati". In caso di annullamento della partita da parte del master a partita in corso, l'informazione di annullamento viene propagata non dal server ma bensì dal client-master.

Il connection handler

Il connection handler si preoccupa di gestire, come dal nome evidenziabile, le connessione con il client. Il connection handler è come l'adviser un task a sé (un thread), questo non appena avviato si mette in attesa sulla server-socket. Ogni volta che un client si connette al server viene un nuovo task chiamato client handler che si preoccuperà di gestire il client in sé. Tra l'avvio del client handler e l'apertura della nuova socket per fa fronte alla richiesta da parte del relativo client, viene eseguito un task che fa in modo che la connessione via socket con il client non si protragga per toppo tempo; il task in questione successivamente ad un tempo di attesa chiude la connessione con il client e il relativo task di handling (il client handler). In questo modo si garantisce che la connessione via socket tra un client e il server non duri più di un tempo prestabilito e che nel caso in cui questa si protragga il tempo prestabilito, la connessione viene chiusa, obbligando così il client a effettuare nuovamente la connessione.

Il client handler

Il client handler non appena avviato si mette in attesa di un messaggio da parte del client. Il client handler gestice due tipologie di messagio:

- Messaggio di tipo "open" per l'apertura di una nuova partita;
- Messaggio di tipo "join" per l'aggregazione di un utente ad una partita precedentemente aperta.

Nel caso in cui il messaggio in questione sia quello di apertura di una nuova partita, questa viene aperta se viene rispettato il vincolo di "massime partite aperte" e se l'indirizzo multicast specificato è diverso da quello che le partite già aperte. Se entrambi i vincoli sono rispettati allora la partita viene aggiunta alla lista di partite aperte e successivamente, facendo uso dell'adviser (vedi apposita sezione) viene inoltrata la lista delle partite disponibili ai vari client. Nel caso considerato, cioè quello in cui i due vincoli sono rispettati, viene inviato al client un messaggio di ACK, questo messaggio funziona da preambolo e fa sì che il client si metta attesa del successivo messaggio, per esempio quello di avvio di partita. Nel caso in cui il primo o il secondo vincolo non venga rispettato viene inoltrato un messaggio con un relativo codice di errore, in questo modo il client visualizzerà la tipologia di errore e potrà quindi correggerlo.
Nel caso in cui il messaggio in questione sia quello di aggregazione ad una partita precedentemente

aperta, allora il client in questione viene aggiunto tra gli utenti partecipanti a quella partita (se la partita è esistente), quindi viene forzato l'invio della lista aggiornata delle partite ai client.

Server closure

Il server può essere terminato in maniera forzata, chiudendo l'applicazione o in tramite il comando di exit. In entrambi i casi tutte le socket e i thread aperti vengono chiusi, questo avviene grazie a due liste di server e thread aperti che nel corso dell'esecuzione del server vengono aggiornate.

Thread e strutture dati

I seguenti sono i thread con relativa descrizione che "possono" essere avviati, a seconda del flusso di esecuzione, dal processo server:

- Adviser: si tratta di un thread implementato dall'omonima classe che implementa due metodi: il sendGameList e il run. Dal secondo metodo si evince che si tratta di un classe che implementa Runnable e che è quindi stata costruita al fine di avviare un thread. La classe Adviser viene instanziata all'interno della classe Server e all'interno della stessa classe Server viene avviato il thread relativo all'Adviser. Nella relativa sezione "Adviser" è specificato il lavoro svolto dalla classe in questione. Il thread creato viene terminato solamente una volta che il server è in corso di terminazione, sia forzato che via comando di "exit".

- ConnectionHandler: si tratta di un thread implementato dall'omonima classe che implementa il solo metodo run da cui si evince che si tratta di una classe che implementa Runnable e che è quindi è stata costruite al fine di avviare un thread. La classe ConnectionHandler viene istanziata all'interno della classe Server e all'interno dello stesa classe Server viene avviato il thread relativo al connectionHandler. Nella relativa sezione "ConnectionHandler" è specificato il lavoro svolto dalla classe in questione. Il thread creato viene terminato nei seguenti casi solamente una volta che il server è in corso di terminazione, sia forzata che via comando di "exit".

- ClientSocketTiming: si tratta di un inner thread instaziato e avviato all'interno del connectionHandler. Questo thread avvia e si sospende nell'attesa di un timeout che segnala l'esaurimento del tempo di durata di vita di una connessione con un client via socket. Scaduto il timeout effettua delle operazioni per la chiusura della relativa socket e rilascia tutte le relative risorse. Nel caso in cui il timeout non scada prima della chiusura del server questo viene terminato una volta che il server è in corso di terminazione. Se invece il timeout scade ma la socket è già stata chiusa perché il la partita ha avuto inizio o il master ha annullato la partita allora lo scadere del timeout non ha alcun effetto se non quello di chiusura del ClientSocketTiming thread.

- ClientHandler: si tratta di un thread implementato dall'omonima classe che implementa il solo metodo run da cui si evince che si tratta di una classe che implementa Runnable e che è quindi è stata costruito al fine di avviare un thread. La classe ClientHandler viene istanziata all'internno della classe ConnectionHandler e all'interno della stessa classe ConnectionHandler viene avviato il thread relativo al ClientHandler. Il thread avviato viene interrotto nel caso in

cui avviene un socket-timeout (quello gestito dal precedente thread) o una volta che il server è in corso di terminazione.

Le principali strutture dati utilizzate del modulo server sono:

- userIdsSocketsMap: Si tratta di una struttura dati di tipo HashMap che ha per chiave l'ID di un utente e per valore la socket istanziata nel momento dell'accettazione della connessione da parte del server. E' stata scelta una struttura dati di tipo HashMap per ragioni efficienza nell'accesso, l'hashmap permette infatti di accedere ad un valore in O(1) il che è il risultato cercato per un oggetto a cui si potrebbe far riferimento spesso.
- clientObjMaps: Si tratta di una struttura dati di tipo HashMap che ha per chiave l'ID di un utente e per valore l'oggetto RMI esposto dal client e a cui il server si è connesso al fine di comunicare la lista delle partite disponibili. E' stata scelta una struttura dati di tipo HashMap per ragioni di efficienza nell'accesso, l'hashmap permette infatti di accedere ad un valore in O(1) il che è il risultato cercato per un oggetto a cui si potrebbe far riferimento spesso.
- serverSockets: Si tratta di una struttura dati di tipo ArrayList che ad ogni apertura di socket da parte del server previa accettazione di una connessione con un client, viene aggiornata aggiornata con l'aggiunta della socket appena creata. Le Socket sono man mano chiuse per via dei vincoli (si rimanda alle specifiche di progetto) e per via delle partite che mano a mano vanno a terminare. Le socket che vengono chiuse vengono rimosse dalla lista, quelle dimenticate (a causa di bug o di situazioni non gestite) vengono terminate una volta che il server è in corso di terminazione.
- serverThreads: Analoga alla struttura precedente per tutti i thread aperti dal processo server.

Altre strutture dati minori sono state utilizzate e state descritte con minor dettaglio per ogni relativa classe nella sezione .Class

Thread concurrency

I metodi contenuti all'interno della classe GameHandler che permettono la rimozione e l'aggiunta di una partita vanno eseguiti in mutua esclusione, per tale motivo alla loro dichiarazione precede "synchronized". Questo è stato reso necessario dal fatto che ClientHandler diversi possono aggiungere/rimuovere partite in maniera concorrente.

Client

Il processo client ha il compito di gestire e dare la possibilità all'utente di interagire con l'applicazione permettendo all'utente di registrarsi o di effettuare il login. Dopo una prima fase ti autenticazione il processo permette all'utente di creare una partita o di aggregarsi ad una già precedentemente creata da un altro utente e quindi, una volta avviata la partita, il processo deve fornire l'interfaccia di gioco.

Non è stata implementata alcuna interfaccia grafica, si lascia ciò a sviluppi futuri. Tutte le funzionalità richieste sono state comunque implementabili e sono fruibili all'utente via terminale.

All'avvio del client viene chiesto l'inserimento di un numero, questo corrisponde all'indice del file di configurazione che deve essere contenuto nella stessa cartella in cui è contenuto l'eseguibile (file .jar) del processo client. Il file di configurazione deve necessariamente essere in formato json, il nome del file deve essere il seguente: "client_conf_?" dove ? corrisponde all'indice numero che andrà ad essere inserito come input per far si che questo possa essere correttamente letto dal processo. Un esempio di file di configurazione è il seguente:

Tutti i campi presenti nell'esempio devono necessariamente essere presenti ed essere avvalorati.

Una volta letto con successo il file di configurazione il flusso di esecuzione passa al menu manager i quali dettagli sono presenti nella relativa sezione.

Menu management

Il primo step del menu managment è quello di visualizzare all'utente i due menù: quello di creazione account/login e quindi, successivamente ad un'autenticazione avvenuta con successo quello di


```
{ "serverSocketAddress":"127.0.0.1",
    "clientAddress":"127.0.0.1",
  "multicastAddress":"224.0.0.1",
    "clientSocketPort":"9099",
    "serverSocketPort":"9098",
     "multicastPort":"9095",
    "RMIClientPort":"35013",
   "RMIServerPort":"35009" }
```

creazione di una partita o join in una partita. Una volta che una partita è stata creata o che la richiesta di join all'interno di una partita è stata inoltrata, il processo client si mette in attesa di un messaggio da parte del server. I messaggi che possono essere ricevuti dal server in questo caso sono:

- Messaggio che indica che non è possibile aprire ulteriori partite.
- Messaggio che indica che la partita a cui si sta cercando di fare parte non esiste.
- Messaggio che indica che la partita è stata annullata.
- Messaggio che indica che la partita può cominciare.

In tutti i casi, fatta eccezione per l'ultimo caso, l'utente viene rimandato al menù di login con il processo nello stato iniziale. Nel caso della ricezione dell'ultimo tipo di messaggio tra quelli elencati il flusso di esecuzione passa al multicast managment, successivamente descritto.

Multicast management

Il multicast managment si occupa della gestione della partita, questa viene portata avanti da messaggi che vengono spediti in multicast. L'indirizzo multicast a cui una relativa partita fa riferimento è definito nel file json di configurazione del client-master che ha aperto la partita. Una volta che il processo entra a far parte del gruppo multicast vengono adottati due comportamenti distinti a seconda se il processo in questione rappresenta un client-guesser o un client-master. Nel caso in cui il processo client in esecuzione rappresenti un client-guesser allora quanto segue è ciò che avviene:

- Viene creato un guesser manager che gestisce l'eventuale uscita da parte del guesser del gioco più implementa un task (thread) che aspetta un carattere dall'utente come tentativo di gioco e aspetta un ack. Nel caso in cui l'ack non viene ricevuto, il task si preoccupa del re-invio del messaggio per 5 volte. A quel punto il gioco viene considerato "morto" e l'utente viene riportato nel menù di login principale. La gestione della ricezione dell'ack di ricezione di un tentativo di gioco non avviene nel task "guesser manager" ma avviene nel task che ha creato quest'ultimo. A seconda se l'ack viene ricevuto o meno una variabile booleana del guesser manager viene settata in maniera mutuamente esclusiva. Il guesser manager aspetterà un timeout, controllerà il valore di tale variabile e agirà di conseguenza inviando nuovamente il tentativo o portando il processo allo stato iniziale nel caso in cui il numero di tentativi ha raggiunto quello massimo. Un messaggio ricevuto da parte di un client-master può non essere un ack, può essere infatti un semplice aggiornamento di stato dovuto ad un tentativo effettuato da un altro client-guesser, può essere un messaggio di fine del gioco dovuto all'esaurimento dei tentativi massimi e può essere un messaggio che notifica il completamento del gioco con successo. Negl'ultimi due casi il flusso di esecuzione viene riportato ad uno stato iniziale e l'utente sarà costretto a rieffettuare il login.

Nel caso in cui il processo client in esecuzione rappresenti un client-master allora quanto segue è ciò che avviene:

- Viene creato un master manager che gestisce l'eventuale uscita da parte del client-master dal gioco e in più implementa un task (thread) che con cadenza di un secondo controlla se l'utente ha inviato il comando di "exit". Nel caso in cui un comando di "exit" è stato inviato,

viene diffusa l'informazione in multicast a tutti gli utenti in modo da informarli che il gioco è da considerasi "morto". Se nessun comando di "exit" è stato inviato, il processo client è costantemente in attesa di un tentativo da parte di qualche client-guesser. Quando questo arriva allora viene inviata risposta in multicast che funzionerà da ack per il client-guesser che ha inviato il tentativo. Le risposte da parte del processo client-master possono essere: un messaggio che informa i client guesser che non sono più a disposizione tentativi (implicitamente informa che l'ultimo tentativo era errato), un messaggio che informa che l'ultimo tentativo corisspondeva ad una lettera contenuta nella keyword (e quindi corrisponderà ad un messaggio di ack per chi ha inviato il tentativo, ad un aggiornamento per gli altri possibili client-guesser) o un messaggio che informa che il gioco è stato completato (il messaggio funziona anche da ack per chi l'utente guesser che ha inviato il tentativo e permetterà quindi di stabilire l'utente vincitore). Una volta che il gioco viene completato o che i tentativi vengono esauriti, il processo relativo al client-master viene riportato allo stato iniziale dove l'utente potrà effettuare nuovamente il login e avviare una nuova partita o aggregarsi ad una partita già aperta. In contemporanea alla ricezione dei tentativi e al controllo dell'eventuale ricezione del comando di chiusura del master, un ulteriore task (thread) verifica che non sia scaduto un tempo massimo consentito per ogni partita. Una volta scaduto questo esegue la funzione dedicata alla chiusura della partita implementata all'interno del master manager.

Thread e strutture dati

I seguenti sono i thread con relativa descrizione che "possono" essere avviati, a seconda del flusso di esecuzione, dal modulo client:

- SocketManager: Si tratta di un thread creato ed avviato all'interno del menù manager che ha come scopo quello di chiudere la socket aperta dal client con il server allo scadere di un timeout. Il thread viene chiuso nel caso venga digitato il comando di "exit" in un qualsiasi stato del processo o nel caso in cui venga ricevuto un messaggio che informa l'utente che la partita aperta a cui si sta cercando di aggregarsi non esiste o un messaggio che informa l'utente che il gioco che si sta cercando di aprire non può essere aperto o un messaggio che indica che l'indirizzo multicast della partita che si sta creando di aprire è esistente o un messaggio che indica che la partita a cui si aveva fatto richiesta di aggregazione è stata annullata o un messaggio che indica che la partita è pronta per essere avviata.

- ExitManager: Si tratta di un thread che viene creato ed avviato all'interno del menu manager che ha come scopo quello di restare in attesa di un'eventuale invio del comando di "exit" da parte dell'utente. Il thread viene chiuso nel caso in cui venga ricevuto un messaggio che indica la non esistenza della partita a cui si sta cercando di aggregarsi o un messagio che indica la non possibilità di aprire una partita o, un messaggio che indica che l'indirizzo multicast che si sta cercando di utilizzare nell'aprire una nuova partita è già in uso o un messaggio che indica che la partita a cui si sta cercando di aggregarsi è stata annullata o un messaggio che indica che la partita che si è creata o a cui ci si è aggregati può iniziare.

- GuesserManager: Si tratta di un thread che viene creato ed avviato all'interno del multicast manager (se l'utente ha assunto carattere di guesser) che ha come scopo quello di attendere l'invio di un carattere da parte dell'utente. Un carattere può essere inviato solamente se è stato ricevuto l'ack del carattere inviato precedentemente (se ne esiste uno). Nel caso in cui

l'ack del carattere inviato precedentemente sia stato correttamente inviato o nel caso in cui si tratta del primo carattere che l'utente invia, il thread si mette in attesa della ricezione dell'ack, fino allo scadere di un timeout. Se il timeout di attesa dell'ack scade e nessun ack è stato ricevuto viene inviato nuovamente lo stesso messaggio con lo stesso carattere digitato dall'utente, e così via per 5 volte. Nel caso in cui l'ack viene ricevuto in uno dei tentativi allora l'utente potrà inviare un nuovo tentativo (nel caso in questione il gioco si considera non concluso), nel caso in cui l'ack non viene mai ricevuto, successivamente ai 5 tentativi il processo clienti ritorna allo stato iniziale considerando la partita come "morta".

- MasterManager: Si tratta di un thread che viene creato ed avviato all'interno del multicast manager (se l'utente ha assuto carattere di master) che ha come scopo quello di attendere l'invio di un comando di "exit" da parte del client-master. Nel qual caso la partita viene annullata e il processo viene riportato nello stato iniziale. Il thread viene chiuso nel caso in cui un comando di "exit" sia stato inviato o nel caso in cui il tempo massimo per partita è scaduto o nel caso in cui non sono disponibili più tentativi di gioco o il gioco è stato completato.

- GameTimeClosure: Si tratta di un thread che viene creato e avviato all'interno del multicast manager che ha come scopo quello di mettersi in attesa dell'eventuale scadenza del tempo massimo per una partita quindi, in caso di scadenza di tale tempo lo stato del processo viene riportato a quello iniziale. Il thread viene chiuso nel caso in cui non sono disponibili più tentativi di gioco o il gioco è stato completato.

Non sono state utilizzate strutture dati di importanza relativa nel processo client per le scelte implementative effettuate. A tal fine si rimanda al codice.

Thread concurrency

Per quanto riguarda la thread concurrency nel processo client, due sono state le situazioni in cui si è reso necessaria la lettura/scrittura in maniera mutuamente esclusiva:

- L'oggetto boolean "AckReceived" instanziato all'interno della classe GuesserManager che permette o meno l'invio di un'ulteriore proposta da parte di un utente: la lettura dell'Ack viene effettuata in un altro thread (quello main) e non nel GuesserManager che si occupa di ritentare l'invio o permettere all'utente di inviare un nuovo tentativo se l'ack è stato ricevuto. Per garantire la corretta lettura/scrittura i metodi getter/setter dell'oggetto all'interno della classe che l'ha instanziato sono stati "marcati" come "synchronized".

- L'oggetto boolean "GameClosure" instanziato all'interno della classe MulticastManager che permette o meno a diversi thread di eseguire delle operazioni solo nel caso in cui la partita non sia in chiusura. Poiché l'oggetto può essere letto e settato da diversi thread, come nel caso precedente per garantire la corretta lettura/scrittura i metodi getter/setter dell'oggetto all'interno della classe che l'ha instanziato sono stati "marcati" come "synchronized".

Common

Durante lo sviluppo del progetto è nata la necessità di creare un modulo comune tra i vari moduli. Questo contiene oltre che a varie costanti utilizzate nel progetto, classi di utilità, come quella che permette il decode e l'encode di un json in stringa, il "common" contiene l'implementazione dei POJO[3]. In particolare i POJO implementati con il relativo significato ed utilizzo sono i seguenti:

- Config: Consiste in una classe pojo di configurazione. Questa contiene tutti i campi necessari a contenere i campi di configurazione del server e del client una volta che questi vengono letti e decodificati da json.
- Game: Consiste in una classe pojo che definisce una partita. Una partita è definita dal suo ID, una lista di utenti che ne fanno parte (client-master incluso), la keywords che man mano viene indovinata, la keywords originale, lo stato del gioco, un numero di utenti minimo per far si che la partita possa partire e il multicast address e port. La classe definisce inoltre i metodi addUser e uncoverCharacter rispettivamente per l'aggiunta di un utente ad un partita e per la "scoperta" di un carattere della keywords.
- Message: Consiste in una classe pojo che definisce un messaggio. Per messaggio si intende quell'oggetto che viene inviato per ogni comunicazione all'interno del gioco tra i processi. Un messaggio viene codicato/decodificato in json (a seconda se questo viene rispettivamente inviato o ricevuto). Un messaggio è definito da un tipo (la classe MessageType di tipo enum definisce tutte le possibili tipologie di messaggio), da un utente (il pojo dell'utente che invia il messaggio), da un carattere (corrisponde al carattere-tentativo che un client può inviare durante una fase di gioco), da una partita (il pojo della partita a cui si fa riferimento), da un master-name (nome dell'utente master a cui si vuol fare riferimento - i.e utilizzato quando si vuol entrare a far parte di una partita aperta di cui per l'appunto si conosce solo il nome dell'utente master), da un ackingID (utilizzando quando il messaggio è una risposta ad un tentativo inviato da parte di un cleint-guesser, corrisponde all'ID dell'utente che ha inviato il tentativo)
- User: Consiste in una classe pojo che definisce un utente. Un utente è definito dall'ID, da un nome, da una password, da un suo indirizzo e porta, da un tipo (un utente può essere di tipo indefinito, master o guesser - tutti i possibili tipi di utenti sono definiti nella classe enumerata UserType) e da una porta RMI.

Tutte le classi pojo in quanto oggetti "inviabili" implementano Serializable.

[3]The term "POJO" initially denoted a Java object which does not follow any of the major Java object models, conventions, or frameworks; nowadays "POJO" may be used as an acronym for "Plain Old JavaScript Object" as well, in which case the term denotes a JavaScript object of similar pedigree

.Class

Di seguito le classi java del progetto hangman divise per modulo e seguite da breve descrizione.

Server

- Adviser: Vedi sezione dedicata.
- ClientHandler: Vedi sezione dedicata.
- ConnectionHandler: Vedi sezione dedicata.
- GameHandler: Si trattra di una classe che ha come scopo quella di gestire le partite aperte (con i relativi metodi di aggiunta a rimozione di partite) e gli utenti appartenenti alle partite. La classe definisce tre metodi synchronized in quanto essendo questi "chiamabili" da diversi thread necessitano di essere eseguiti in maniera mutuamente esclusiva. La struttura dati utilizzata e definita nella classe è un'hashmap che come chiave utilizza l'ID di una partita e come valore il pojo.
- RmiServerImpl: Si tratta della class che implementa i metodi esposti tramite RMI al client. Per questione di coerenza in realtà i metodi sono implementati all'interno dell'UserManager, il relativo oggetto è "passato" come parametro al momento dell'istanziazione dell'oggetto RmiServerImpl quindi i relativi metodi esposti dall'UserManager (createAccount, login e logout) sono chiamati quando gli omonimi della classe RmiServerImpl vengono chiamati.
- Server: Si tratta della classe "main" che instanzia l'oggetto server.
- UserManager: Si tratta di una classe che ha come scopo quella di gestire la parte utente a livello server. La classe contiene metodi per la validazione dell'utente, della password per la creazione di un utente e per il login. La classe definisce inoltre un metodo per la connessione con RMI che il relativo client espone.

Client

- Client: Si tratta della classe "main" che instanzia l'oggetto client.
- GameManager: Si tratta di una classe che definisce i seguenti tre metodi: createGame, joinGame e logout.
- GuesserManager: Si tratta di una classe che definisce il medesimo oggetto che implementa Runnable e che sarà quindi eseguito come un threadin una fase di esecuzione del client. Lo scopo funzionale è quello definito nella sezione Client alla sottosezione Thread -guesserManager.
- MasterManager: Si tratta di una classe che definisce il medesimo oggetto che implementa Runnable e che sarà quindi eseguito come thread in una fase di esecuzione del client. Lo scopo funzionale è quello definito nella sezione Client alla sottosezione Thread - masterManager.
- MenuManager: Vedi sezione dedicata.
- MulticastManager: Vedi sezione dedicata.

- RMIClientImpl: Si tratta della classe che implementa i metodi esposti tramite RMI al server. Nel caso in questine il metodo è sono uno, sendAvaiableGames. Una volta ricevuta l'array di partite disponibili allora per ogni partita viene "estratto" l'utente master dal pojo della partita quindi viene stampato partita dopo partita l'utente master di ogni partita all'utente guesser al fine di dare la possibilità al client guesser di inviare un comando di aggregazione ad una specifica partita gestita da uno specifico client-master.
- UserManager:

Common

Il common contiene oltre ai POJO già descritti nella relativa sezione "Common" anche delle classi di utilità:

- JsonUtils: Si tratta di una classe statica che definisce delle funzioni per la manipolazione dei JSON (encodeJSON, decodeJSON).
- Utils: Si tratta di una classe statica che definisce la funzione per il calcolo dell'md5. Questa funzione è utilizzata per il calcolo dell'ID di un utente (l'id di un utente corrisponde all'md5 dell'username e per la password dell'utente. La password non è mai nè salvata nè inviata in chiaro) e per il calcolo dell'ID di una partita)l'id di una partita corrisponde all'md5 del nome utente di colui che ha creato la partita concatenato all'unix epoch millisec.).

Test ed esecuzione

Il progetto consegnato non è da considerarsi finale o per usi diversi da quelli didattici. Per tal motivo il progetto non è da considerarsi esente bug o problemi di varia natura. Tutti i test sono stati effettuati in ambiente di debug dell'IDE e con i file di configurazione allegati. Il progetto è funzionante nelle funzionalità che seguono:

- Apertura del server e attesa della connessione di un client;
- Apertura di un client, creazione account/login/creazione partita/join in una partita

I test sono stati effettuati con le seguenti partite:

- Partita aperta che necessità di due giocatori
- Partita aperta che necessità di tre giocatori.

Nota bene che quando si definisce il numero di giocatori quello include anche il master, una partita con tre giocatori sarà quindi una partita tra due guesser più il master.

Una volta che il progetto è stato ritenuto sufficientemente funzionante sono stati eseguiti gli artifact server e client al fine di eseguire il progetto da terminale e quindi non in debug mode. Sono stati realizzati due video di test rispettivamente uno che descrive lo svolgimento di una partita con un guesser e un master e un secondo video che descrive lo svolgimento di una partita con due guesser e un master. I video sono presenti nel seguente canale: https://www.youtube.com/user/simonesalvo101.

Riferimenti

- JavaDoc 7: http://docs.oracle.com/javase/7/docs/api/
- Gson: https://github.com/google/gson
- Lombok: https://projectlombok.org/
- Jasypt: http://www.jasypt.org/