UNIVERSITÀ DI PISA

CdL in Informatica - Laboratorio di Reti, Corso A

WINSOME: a re**W**ard**IN**g **SO**cial **ME**dia

Progetto di Fine Corso A.A. 2021/22

Simone Marzeddu

INTRODUZIONE - Uno sguardo generale all'architettura di Winsome

Il progetto conclusivo del corso ha richiesto lo sviluppo di un Social Network, dal nome Winsome, sia nell'implementazione lato Server che in quella lato Client.

Il Client, che rispetta le caratteristiche di "thin client" studiate a lezione, consente ad un utente di operare su Winsome attraverso la digitazione di comandi run-time; a seguito di queste richieste, che raggiungono il Server attraverso l'utilizzo di TCP o RMI secondo quanto richiesto dalla specifica, questo esegue determinate funzioni nel tentativo di soddisfare i desideri dell'utente, comunicando infine i risultati delle sue operazioni.

In particolare, con riferimento a tutte le connessioni TCP, il Server dedicherà ad ogni Client connesso uno specifico thread (si veda la sezione dedicata ai Threads per ulteriori approfondimenti) così da poter univocamente identificare lo stato di login di un determinato utente e da gestire indipendentemente dagli altri le richieste di quest'ultimo.

Alcune delle personali scelte progettuali riguardano proprio la gestione degli accessi degli utenti al Social:

Come indicato, il login deve essere eseguito da ogni utente prima che questo possa avere delle interazioni importanti con Winsome, la scelta progettuale è stata quella di far sì che il sistema tenga traccia dell'indirizzo univoco di ogni determinato socket associato ad un Client, consentendo così agli utenti di essere connessi alla rete sociale con al massimo un solo dispositivo o pagina per volta.

Conservando questa informazione anche al livello del "worker" assegnato ad un determinato utente, il sistema riesce ad impedire il login contemporaneo a più accounts da parte di un Client.

Per quanto riguarda invece i sistemi di notifica ammessi dal Client, intendendo così sia il sistema di RMI Callback utilizzato per la raccolta di informazioni sugli utenti "followers" lato Client che le notifiche per gli avvenuti calcoli delle ricompense inviati con l'ausilio del Multicast, sono state adottate delle implementazioni particolari:

- Al fine di ottenere uno stato consistente lato Client per quanto concerne la raccolta dei followers, il Server invocherà il metodo "consistent_state_followers" da remoto, aggiornando così le informazioni di cui il primo dispone, ad ogni nuova registrazione per la Callback (operazione che il Client eseguirà attraverso RMI ad ogni nuovo login).

- Spostando invece l'attenzione sul sistema di notifiche (uguali per tutti gli utenti e ottenute dai Client attraverso il Multicast) si è preferito in fase implementativa offrire all'utente il comando speciale "show notifications" così da permettere a questo la visualizzazione istantanea di tutte le notifiche ricevute dall'avvio del software al momento della richiesta.

- Sempre sul punto precedente, anche prendendo in considerazione il fatto che le notifiche di cui sopra sono totalmente pubbliche e generali, il comando "show notifications" non richiede il login di uno specifico utente; il Client raccoglierà infatti tutte le notifiche di questo tipo ricevute durante il suo periodo di attività.

Sorvolando ora su tutto ciò che riguarda la gestione delle varie connessioni ed interazioni Client/Server è opportuno concentrarsi sul vero e proprio nucleo di Winsome:

La classe "Social Network" raccoglie al suo interno le implementazioni dei metodi principali dell'intero sistema, questi, sfruttando anche quanto offerto dalle classi "Post" e "User", permettono una completa e sicura gestione da parte di thread esterni delle due strutture dati "winsome_posts" e "winsome_users" non che dei loro contenuti: Ogni utente di Winsome è infatti rappresentato da una specifica istanza di "User" così come ogni post pubblicato nel social ha la sua rappresentazione in un'istanza di "Post".

Le scelte progettuali adottate in questi frangenti sono state le seguenti:

- In ogni istanza di "User" non viene mai memorizzata alcuna password in chiaro, questo poiché è stato ritenuto necessario proteggere i dati degli utenti di Winsome da eventuali attacchi che il Server o in generale i file a questo relativi potrebbero subire; per consentire dunque di poter autenticare correttamente un determinato utente U si confronta il risultato della funzione hash "SHA-256" applicata alla stringa "password inserita da U + seed" con il valore ottenuto allo stesso modo durante la registrazione di U (Si noti che il seed viene generato casualmente al momento della registrazione di un utente per poi venire conservato all'interno della sua istanza di "User").

- Le valutazioni ed i commenti che possono essere associati ad ogni post in Winsome si ritrovano nel sistema con la classe "Feedback" e la sua estensione gerarchica "Comment", le istanze di queste sono mantenute all'interno di opportune strutture dati incapsulate in ogni post e sono utilizzate in ausilio a quanto offerto dalla classe "Post" stessa.

- Sempre nella classe “Post” si ritrovano alcune variabili apparentemente superflue come “resettable_rates_sum”, “resettable_new_commentators” e “rewardable_users”, queste sono utilizzate come un’alternativa a quella che sarebbe altrimenti stata la visita di ogni singolo oggetto di tipo “Feedback” o “Comment” durante il calcolo delle ricompense relative ad un dato post e permettono di velocizzare quest’ultimo.

Si analizzeranno di seguito i vari threads che Client e Server attivano e le scelte implementative in termini di strutture dati utilizzate e di sistemi adottati per garantire la sicurezza del multithreading nell’ottica di preservare comunque delle performance ottimali del software.

CONCORRENZA - Thread attivati dal processo **Client** e strutture dati utilizzate

Il processo Client inizia la sua esecuzione con il codice della classe “ThinClientMain”, contenente appunto il metodo “main” del processo, per prima cosa vengono effettuati dei controlli sulla presenza di argomenti passati in input al programma al suo avvio e qualora questo controllo vada a buon fine il processo prosegue con la chiamata al metodo “setProperty” impostando le politiche di sicurezza del sistema con quanto contenuto nel file “MyGrantAllPolicy.policy” e selezionando come “security manager” un nuovo oggetto di tipo “RMISecurityManager”.

Il primo argomento ceduto all’avvio del programma viene quindi utilizzato per la costruzione di un oggetto di tipo “URL” e nel caso in cui un secondo argomento sia stato individuato, questo verrà interpretato come percorso del file di configurazione per il Client.

Solo a questo punto la classe “LoadClient” viene caricata con il metodo “loadClass” statico della classe “RMIClassLoader” grazie anche all’utilizzo del parametro di tipo “URL” prima nominato, da questa classe appena caricata verrà invocato il costruttore (a cui sarà ceduto l’eventuale percorso del file di configurazione) per la costruzione di un’istanza di “LoadClient” dalla quale verrà invocato il metodo “run”.

Dopo l’iniziale lettura del file di configurazione l’attività vera e propria del Client potrà prendere inizio (si noti che delle impostazioni standard saranno adottate nel caso in cui non venga rilevato un file di configurazione appropriato), a questo punto sarà ottenuta, grazie al metodo “lookup” sul “Registry” del Server, l’interfaccia “ServerRemoteInterface” con cui potranno essere invocati i metodi “register” (per la registrazione di un utente su Winsome), “registerForCallback” (che permetterà al Client di iscriversi al sistema di notifica per il mantenimento della lista dei followers di un utente lato Client) ed “unregisterForCallback” (grazie al quale il Client si disiscriverà dal servizio precedente).

Nella classe “LoadClient” risiede dunque il vero cuore pulsante di tutto il processo Client, è qui infatti che il thread principale utilizzerà uno “Scanner” per la lettura dei comandi dell’utente da tastiera , procedendo con il puntuale controllo della loro sintassi e con l’invio delle richieste al Server attraverso un “DataOutputStream” e lettura dei risultati in arrivo dal Server da un oggetto di tipo “DataInputStream” (entrambi gli “stream” sono ovviamente costruiti a partire dal “Socket” per la connessione TCP con il Server).

L’implementazione attraverso l’uso delle classi “DataInputStream” e “DataOutputStream” si è rivelata indispensabile a causa di alcune incompatibilità sorte lato Client relativamente all’uso di “Scanner” e “stream” di altra forma come le versioni “Buffered”.

Il secondo thread rimane invece in ascolto sul “MulticastSocket” in attesa di ricevere i pacchetti che il Server invia ad ogni nuovo calcolo delle ricompense per poi archiviare i dati raccolti, insieme alla relativa data e ora di arrivo (ottenute attraverso la classe “Instant”), in una cronologia locale all’oggetto istanza della classe “ClientMulticastNotificationTask” implementante l’interfaccia “Runnable” ed al partire dal quale il thread è costruito.

Si è rivelato dunque necessario sincronizzare gli interventi alla cronologia appena nominata, implementata come un oggetto di tipo “StringBuilder” di nome “history”, in vista del fatto che

questa è soggetta alla concorrenza di due threads; ciò è stato raggiunto attraverso l'incapsulamento di tutte le sezioni critiche in blocchi sincronizzati su questa.

Sempre sul lato Client e sul tema della sincronizzazione, la classe "ClientNotify" offre l'implementazione di metodi (alcuni dei quali anche invocati da remoto tramite RMI dal Server) utili alla manipolazione di un oggetto locale di tipo "LinkedList" il quale raccoglie informazioni relative ai "followers" dell'utente attualmente autenticato sul Client; la sincronizzazione all'interno di questa classe è stata ottenuta dichiarando "synchronized" i metodi della stessa.

CONCORRENZA - Thread attivati dal processo **Server** e strutture dati utilizzate

Il processo Server presenta invece una struttura decisamente più articolata, generando un ampio ventaglio di thread utili sia alla soddisfazione delle richieste dei Clients sia all'esecuzione di tutte le operazioni previste invece come "background" della sua attività:

Il thread principale (il cui codice risiede nella classe "ServerMain" da cui il processo inizia la sua esecuzione) prevede un parsing del file di configurazione (il cui percorso è un argomento passato all'avvio del software) per poi procedere con la deserializzazione dei files Json contenenti i backup della rete sociale, ciò attraverso l'invocazione del metodo "deserializeWinsome" della classe "SocialNetwork", e con la creazione di un "Registry" per permettere l'uso del servizio RMI da parte del Client; sarà poi questo thread ad occuparsi della generazione di altri tre thread portanti del processo oltre che della gestione del segnale SIGINT (che comporterà un ultimo backup dello stato della rete sociale prima della terminazione del Server).

Il thread di backup (costruito a partire dall'oggetto "ServerBackupTask" implementante l'interfaccia "Runnable") è uno dei thread generati a partire dal thread principale e si occupa della gestione, indipendente dal resto del sistema, dei backup automatici di Winsome ottenuti con la serializzazione Json delle istanze di "Post" e "User" all'interno dell'oggetto di tipo "SocialNetwork".

Questa attività consiste nell'invocazione ciclica del metodo "serializeWinsome", offerto dalla classe di sopra indicata, che si occupa della serializzazione in stringhe Json degli oggetti presenti nelle variabili di istanza "winsome_users" e "winsome_posts", due oggetti di tipo "ConcurrentHashMap".

Non è risultata necessaria l'implementazione di un meccanismo di sincronizzazione in questo frangente poiché tutte le strutture dati di tipo "ConcurrentHashMap" offrono un pieno supporto della concorrenza.

Il thread per il calcolo delle ricompense (costruito a partire dall'oggetto "ServerRewardsTask" implementante l'interfaccia "Runnable") inizia il suo flusso di esecuzione nella generazione dal thread principale, procede poi con la partecipazione al Multicast come publisher per poi avviare l'invocazione ciclica del metodo "updateRewards" offerto dal social con relativo invio dei pacchetti di notifica sul canale Multicast.

Il metodo "updateRewards" consiste nel calcolo delle ricompense (conforme alla formula indicata nel testo del progetto) per ogni singolo post pubblicato su Winsome ed in un aggiornamento dei portafogli per ogni utente meritevole della ricompensa nelle proporzioni stabilite dal file di configurazione.

Per le stesse motivazioni del punto precedente non è stata richiesta particolare attenzione nei confronti della sincronizzazione al livello dei metodi offerti da "SocialNetwork"; tuttavia, lo stesso discorso non vale invece per il codice dei metodi ausiliari implementati sulle classi "Post" e "User" invocati dal metodo "updateRewards" sopra citato.

Nel caso della classe "Post" è risultato essenziale l'ottenimento della mutua esclusione sulle variabili locali "resettable_new_commentators", "reward_calculation_iter_num", "resettable_rates_sum" e "rewardable_users" a causa della forte concorrenza a cui sono soggette.

Trattandosi di parametri esclusivamente legati al calcolo delle ricompense, la scelta implementativa per la sincronizzazione è ricaduta sull'utilizzo di un oggetto di tipo "ReentrantLock" denominato "rewards_calculation_lock" il cui metodo "lock" viene invocato all'inizio di ogni sezione critica coinvolgente le variabili locali di cui sopra (ovviamente incapsulato in un blocco "try" seguito dal rispettivo blocco "finally" per garantire l'invocazione del metodo "unlock").

Un discorso simile vale quindi anche per la classe "User" in cui gli accessi alle variabili "wallet_wincoin" e "transaction_history" richiedono una previa chiamata del metodo "lock" dell'oggetto di tipo "ReentrantLock" denominato "wallet_lock" presente in ogni istanza utente, anche in questo caso il metodo viene invocato all'interno di un blocco "try" seguito dal corrispettivo blocco "finally" in cui si garantisce la chiamata del metodo "unlock" sullo stesso oggetto.

Il thread per l'amministrazione delle connessioni TCP (costruito a partire dall'oggetto di tipo "ServerTCPAdmin" implementante l'interfaccia "Runnable") è l'ultimo dei tre thread generati dal thread principale, inizia la sua esecuzione con l'apertura di un "ServerSocket" e procede invocando ciclicamente il metodo "accept" di questo in attesa di nuove connessioni.

Come già è stato accennato nella sezione precedente ogni Client connesso al Server viene servito da un "thread worker" dedicato, il thread amministratore delle connessioni TCP genererà dunque uno di questi in occasione di ogni nuova "accept" terminata con successo, affidando ad una "CachedThreadPool" un oggetto di tipo "ServerTCPTask" implementante l'interfaccia "Runnable".

L'ultima tipologia di thread generati lato Server sono proprio i sopra definiti "thread workers", generati in seguito ad ogni "accept" andata a buon fine nel thread precedentemente descritto e costruiti a partire dalla classe "ServerTCPTask" di cui sopra.

Sono questi thread a rappresentare l'interfaccia di dialogo tra Client e Server attraverso l'utilizzo di una connessione TCP e in particolare di oggetti di tipo "DataInputStream" e "DataOutputStream" dai quali leggono e scrivono con l'uso dei metodi "writeUTF" e "readUTF" le richieste provenienti dal Socket del Client assegnatogli al momento della costruzione della "task" ed i relativi output restituiti dalle chiamate dei metodi nell'oggetto di tipo "SocialNetwork" denominato "winsome" anch'esso ottenuto come riferimento dalla costruzione della "task".

L'attività di questi thread inizia con la comunicazione al Client di porta e indirizzo per la ricezione di notifiche tramite il servizio di Multicast, è stata infatti questa la scelta implementativa per la trasmissione di queste informazioni ai vari Client, così che questi possano poi proseguire con la generazione del thread dedicato a questa comunicazione già analizzato nella sezione dedicata ai thread generati lato Client.

Dopo questo primo scambio di informazioni il thread inizierà ad attendere le richieste del Client individuando le parole chiave dei comandi ed i relativi argomenti inviati come stringhe da questo, che grazie ai controlli presenti lato Client saranno assicurati essere ben formati.

All'interno degli oggetti istanze della classe "ServerTCPTask" sono incapsulate anche delle informazioni inerenti allo stato di login di un determinato utente: "who_are_they" è una stringa contenente il risultato della chiamata al metodo "getRemoteSocketAdress" sul Socket del Client e ne rappresenterà dunque un identificativo univoco per l'istanza, "actual_username" registrerà invece l'username dell'utente attualmente autenticato sul Client in seguito al comando "login <username> <password>".

Queste informazioni hanno lo scopo di gestire i permessi di interazione dei vari Client con i dati di Winsome e garantiscono inoltre (in combinazione all'implementazione delle altre classi lato Server) che un utente possa effettuare il "login" attraverso al più un Client contemporaneamente.

Un ultimo dettaglio sulla classe "ServerTCPTask" nonché sui thread costruiti a partire da suoi oggetti istanza riguarda le scritture sul "DataOutputStream" verso il Client, infatti, i risultati delle operazioni richieste dagli utenti potrebbero potenzialmente consistere in stringhe di dimensioni non trascurabili, per questo, essendo noto dalla documentazione che le stringhe scrivibili con il metodo "writeUTF" in questo genere di "stream" possono avere una lunghezza massima di 65535

Bytes, è stato indispensabile segmentare le stringhe output di richieste come "list users", "list following", "wallet" e "wallet btc" attraverso una scrittura mediata dal metodo "writeUTF_segmented" implementato all'interno della classe.

Un ultimo appunto sulla concorrenza dei thread di questa tipologia riguarda la classe "User" ed in particolare i suoi metodi "login", "logout" e "not_authentic" i quali sono spesso invocati dai metodi della classe "SocialNetwork" durante i controlli antecedenti all'elaborazione di ogni richiesta o condividono comunque delle sezioni critiche con queste operazioni (ovvero gli accessi alla variabile "login-address" contenente una copia dell'indirizzo remoto del socket relativo al Client con cui l'utente si è autenticato sul Server) per queste ragioni questi metodi sono stati dichiarati "synchronized".

Poiché anche le istanze di "LinkedList" presenti in ogni oggetto di tipo "User" subiscono frequenti accessi concorrenti essi sono opportunamente incapsulati in blocchi dichiarati "synchronized" su queste.

La prossima sezione tratterà singolarmente ogni classe costituente il progetto "Winsome", avendo però già abbondantemente analizzato alcune classi quali: "ServerMain", "ServerBackupTask", "ServerRewardsTask", "ServerTCPAdmin", "ServerTCPTask", "ThinClientMain", "LoadClient", "ClientNotify" e "ClientMulticastNotificationTask", queste non riceveranno una trattazione dedicata.

Altre classi come "User", "Post" e "SocialNetwork" subiranno un'analisi meno approfondita, ma volta a completare le informazioni che già sono state fornite su queste.

LE CLASSI DI WINSOME – una descrizione sintetica delle classi implementate

- La classe "SocialNetwork" incapsula, come già ribadito dalle precedenti sezioni, l'implementazione necessaria per svolgere le operazioni richieste dai vari Client attraverso i comandi degli utenti che qui ritrovano usualmente un metodo direttamente associato ed invocabile dai "thread workers".
  Oltre all' informazione sulla percentuale di ricompensa spettante al creatore di un dato post (ricevuta attraverso il costruttore e prima eventualmente dal file di configurazione del Server) le altre variabili locali presentate dalla classe sono le "ConcurrentHashMap" denominate "winsome_users", "winsome_posts" e "callbacks_registrations": le prime due rappresentano semplicemente lo stato di Winsome in termini rispettivamente di utenti iscritti e post pubblicati sul social mentre la terza struttura dati mantiene invece i riferimenti alle interfacce "ClientNotifyInterface" dei vari Clients registratisi per il servizio di "RMI Callback" e distinguibili dall'"username" dell' utente in essi autenticato ed utilizzato in questo oggetto come chiave (motivo per cui un solo utente per volta può effettuare correttamente il login in un Client).

- La classe "Post" rappresenta i post pubblicati all'interno di Winsome e raccoglie informazioni sui feedback, i commenti e valori ausiliari per il calcolo delle ricompense relativi a questi.
  Questa classe implementa al suo interno due differenti costruttori di cui uno è volto alla generazione di nuovi post originali mentre l'altro consente la ricostruzione di un'istanza valida dopo la deserializzazione Json di un'oggetto di questo tipo.
  Un istanza di "Post" è univocamente identificata dal suo "post_id" ed incapsula al suo interno due stringhe rappresentanti titolo e contenuto del post (rispettivamente "title" e "body"), si noti che i contenuti dei post vengono formattati durante la costruzione.
  Oltre a quanto già esposto nelle sezioni precedenti la classe "Post" implementa anche una vasta gamma di metodi volti all'interazione ed all'ottenimento dei contenuti di "likes" "dislikes" e "comments", strutture dati improntate al supporto per il multithreading contenenti feedbacks e commenti che il post potrà ricevere durante la sua vita su Winsome.

- La classe “Feedback” e la sua estensione “Comment” rappresentano le valutazioni ed i commenti che i post di Winsome possono ricevere, nella classe padre ritroviamo esclusivamente un “timestamp” comprendente data ed ora della creazione di una sua istanza ed una stringa “author” che ne identifica l’autore; la classe “Comment” si limita ad estendere la precedente con l’incapsulamento di una stringa rappresentante il testo del commento ed un metodo per estrarre questo da una sua istanza, sulla falsa riga dei metodi ereditati da “Feedback” che offrono un’implementazione analoga verso i valori precedentemente analizzati.

- Le classi “PostRewardsCalculationOutput” e “PostRewardsState” si limitano ad incapsulare dei dati e metodi “getter” per questi, ritrovano la loro utilità esclusivamente nell’ offrire la possibilità di ottenere riferimenti a molteplici informazioni in un unico output per i metodi “getRewardsState” e “calculateRawRewards” nella classe “Post”.

- La classe “User” rappresenta un utente di Winsome ed implementa al suo interno due differenti costruttori di cui uno è volto alla generazione di nuove istanze dedicate ad utenti appena registrati mentre l’altro consente la ricostruzione di un’istanza valida dopo la deserializzazione Json di un’oggetto di questo tipo.
  Oltre alla presenza di metodi “getter” specifici per l’ottenimento da parte della classe “SocialNetwork” di stringhe particolarmente formattate nella classe “User” si ritrovano anche numerosi altri metodi simili utilizzati per ottenere esternamente le strutture dati o più in generale gli oggetti da serializzare nel momento in cui questo processo coinvolgerà l’oggetto corrente.
  Ogni utente di Winsome è univocamente identificato dal suo “username” che una volta scelto è immutabile ed appunto univoco.
  Gli oggetti incapsulati di tipo “ConcurrentHashMap” denominati “blog” e “feed” sono le strutture dati in cui i riferimenti ai “Post” appartenenti agli omonimi concetti di Winsome relativi ad un dato utente sono memorizzati, la classe offre svariati metodi per aggiornare i contenuti di queste strutture in totale sicurezza dal punto di vista della concorrenza.
  Due oggetti di tipo “LinkedList” sono utilizzati nelle istanze di questa classe per conservare gli usernames degli utenti seguiti e che seguono gli utenti da esse rappresentati, prendono infatti il nome di “followed” e “followers”, gli aspetti relativi alla gestione della concorrenza su queste strutture dati ed al trattamento dei dati sensibili degli utenti sono già stati trattati nelle sezioni precedenti.

- La classe “Hash” offre semplicemente due metodi statici utili al calcolo del valore “SHA-256” di una stringa e la sua conversione in esadecimale.

MANUALE – istruzioni su come compilare, eseguire ed utilizzare il software

La cartella principale del progetto contiene al suo interno cinque sottocartelle:

- La cartella “bck” è stata intesa come il luogo in cui il Server andrà a depositare i files contenenti la serializzazione di Winsome, dal primo avvio di questo, infatti, questa cartella conterrà i files “winsome_posts” e “winsome_users”,
- La cartella “config” è stata pensata come contenitore per i files di configurazione, sia per Server che per Client; tuttavia, il percorso del file di configurazione può essere liberamente selezionato dall’utente in fase di esecuzione dei processi.
- La cartella “jar” contiene i files “Winsome_Client.jar” e “Winsome_Server.jar”, è possibile semplicemente eseguire questi files con gli opportuni parametri (le istruzioni sull’esecuzione saranno riportate successivamente) per utilizzare il sistema.
- La cartella “lib” contiene il file “gson-2.8.5.jar” ovvero il file “.jar” della libreria utilizzata per la serializzazione Json del social.

- La cartella "src" contiene tutti i files sorgente in formato ".java", il file "MyGrantAllPolicy.policy" per impostare le politiche di sicurezza ed i files ".mf" utilizzati per la produzione dei files ".jar".

Per operare la compilazione del codice è sufficiente avviare la shell, posizionarsi nella cartella "src" del progetto e digitare il comando:

- javac -cp .;../lib/gson-2.8.5.jar *.java (Windows)
- javac -cp .:../lib/gson-2.8.5.jar *.java (Linux e MacOS)

A questo punto è possibile eseguire il Server con il comando:

- java -cp .;../lib/gson-2.8.5.jar ServerMain ..\config\Server_config.txt (Windows)
- java -cp .:../lib/gson-2.8.5.jar ServerMain ..\config\Server_config.txt (Linux e MacOS)

Qualora si volesse eseguire il Server utilizzando il file "Winsome_Server.jar" della cartella "jar" sarà sufficiente posizionarsi su questa e digitare il comando:

- java -jar Winsome_Server.jar ..\config\Server_config.txt

Ed è similmente possibile eseguire il Client con il comando:

- java -cp .;../lib/gson-2.8.5.jar ThinClientMain file:/. ..\config\Client_config.txt (Windows)
- java -cp .:../lib/gson-2.8.5.jar ThinClientMain file:/. ..\config\Client_config.txt (Linux e MacOS)

Qualora si volesse eseguire il Client utilizzando il file "Winsome_Client.jar" della cartella "jar" sarà sufficiente posizionarsi su questa e digitare il comando:

- java -jar Winsome_Client.jar file:/. ..\config\Client_config.txt

NOTA: L'argomento passato all'esecuzione del Server "..\config\Server_config.txt", così come quello passato all'esecuzione del Client "..\config\Client_config.txt", è il percorso relativo rispetto alla cartella di esecuzione del file di configurazione scelto per il software.

NOTA: L'argomento passato all'esecuzione del Client "file://." è la stringa utilizzata per la costruzione dell' oggetto "URL" utilizzato per la ricerca della classe "LoadClient" all'interno del "thin client".

Seguirà ora l'esposizione di una guida all'utilizzo del software Client per l'interazione con Winsome.

In seguito alla comparsa del messaggio di benvenuto sarà possibile immettere i propri comandi, nel caso in cui si verificasse un qualunque tipo di errore il sistema provvederà alla notifica di questo, evidenziandone le cause, così come restituirà un feedback di successo nel caso in cui l'operazione sia andata a buon fine.

Prima di elencare i comandi ammissibili si noti che tutti gli argomenti a questi relativi dovranno essere raccolti tra simboli "<" e ">" e separati da spazi proprio come mostrato in seguito:

- l'immissione del comando "help" determinerà la stampa a schermo di un elenco dettagliato di tutti i comandi disponibili e la loro sintassi oltre che di una breve descrizione,

- l'immissione del comando "show notifications" risulterà nella stampa a schermo di tutte le notifiche inerenti ai calcoli delle ricompense di Winsome,

- l'immissione del comando "register <username> <password> <tags>" porterà alla registrazione di un nuovo utente su Winsome a patto che il suo "username" non sia ancora mai stato selezionato ed i tags consistano in un massimo di cinque parole separate da spazi,

- l'immissione del comando "login <username> <password>" consentirà all'utente registrato con il nome "username" di accedere alla propria area privata, con l'ottenimento di tutti i permessi che ne conseguono.

- L'immissione del comando "quit" porterà alla terminazione del processo Client preceduta da un messaggio di arrivederci.

ATTENZIONE: Tutti i prossimi comandi richiedono, per poter essere eseguiti da un determinato utente, che questo abbia correttamente completato una precedente operazione di "login".

- L'immissione del comando "logout" consentirà un uscita sicura dal proprio account, sarà indispensabile quindi eseguire un nuovo "login" per ottenere i permessi relativi ad un eventuale utente desideroso di operare su Winsome,

- L'immissione del comando "list users" permetterà all'utente di leggere a schermo i nomi degli utenti che condividono almeno uno dei tags scelti al momento della registrazione di questo.

- L'immissione del comando "follow <username>" consentirà all'utente corrente di seguire l'utente con username "username", in questo modo sarà possibile visualizzare i post di quest'ultimo nel proprio feed,

- L'immissione del comando "unfollow <username>" consentirà all'utente corrente di rimuovere il "follow" eseguito in passato nei confronti dell'utente con username "username",

- L'immissione del comando "list followers" consentirà all'utente corrente di visualizzare una lista contenente tutti gli utenti che hanno operato un "follow" su questo.

- L'immissione del comando "list following" consentirà all'utente corrente di visualizzare una lista degli utenti da lui seguiti attraverso il comando "follow",

- L'immissione del comando "post <title> <content>" permetterà all'utente corrente di pubblicare un post su Winsome con il titolo "title" ed il contenuto "content" che potranno avere un'estensione massima di rispettivamente venti e cinquecento caratteri,

- L'immissione del comando "rate <post_id> <vote>" permetterà all'utente corrente di esprimere una valutazione "vote" (con valore esclusivamente uguale a +1 o -1) a patto che il post con id "post_id" sia presente sul proprio feed e non sia stato pubblicato dall'utente stesso,
- L'immissione del comando "comment <post_id> <content>" permetterà all'utente corrente di pubblicare un commento dal contenuto uguale a "content" verso il post con id "post_id" a patto che questo sia presente sul proprio feed e non sia stato pubblicato dall'utente stesso,

- L'immissione del comando "show post <post_id>" consentirà la visualizzazione a schermo del contenuto del post identificato dall'id "post_id" a patto che questo sia un valore valido, saranno inoltre visualizzate altre informazioni come, ad esempio, valutazioni e commenti ricevuti dal post,

- L'immissione del comando "delete <post_id>" rimuoverà il post identificato dall'id "post_id" da Winsome, l'operazione sarà consentita esclusivamente a patto che il post sia stato pubblicato sul social network dall'utente corrente,

- L'immissione del comando "rewin <post_id>" permetterà all'utente corrente di condividere un post presente all'interno del proprio feed così da renderlo disponibile anche all'interno

del proprio blog (l'autore originale avrà tuttavia pieni diritti sulla cancellazione del post e sulle ricompense autoriali),

- L'immissione del comando "show feed" consentirà all'utente corrente di visualizzare una lista di tutti i post pubblicati dagli utenti da questo seguiti (attraverso il comando "follow"),

- L'immissione del comando "blog" consentirà all'utente corrente di visualizzare la lista di tutti post da egli pubblicati nel corso dei suoi utilizzi di Winsome,

- L'immissione del comando "wallet" consentirà all'utente corrente di visualizzare il valore in Wincoins del suo portafoglio insieme ad un riassunto di tutti i guadagni mai ricevuti,

- L'immissione del comando "wallet btc" consentirà all'utente corrente di visualizzare il valore in Bitcoins del suo portafoglio insieme ad un riassunto di tutti i guadagni mai ricevuti.