RELAZIONE PROGETTO DI RETI

GIOVANNI SALINITRO MATRICOLA:560226

Premesse

Il progetto WINSOME è stato svolto usando l'IDE Visual Studio Code su ambiente Windows. I comandi per la compilazione e l'esecuzione del progetto saranno quindi proprio del prompt dei comandi di Windows. Il progetto sarà già compilato ma, spiegando successivamente le istruzioni per la compilazione, può essere nuovamente compilato ed eseguito.

Descrizione generale dell'architettura complessiva del sistema

Il progetto consiste di due classi Main: ServerMain.java, la cui esecuzione fa partire il server, e ClientMain.java, la cui esecuzione farà partire il client per interagire col server.

Lato Server:

All'esecuzione di ServerMain.java partirà come detto il server, il quale leggerà i parametri scritti nel file di configurazione testuale "server.properties" (che si trova nella cartella "config",dentro la cartella "src") e creerà un'istanza della classe WinsomeServer; questa è la classe dove sono implementati tutti i vari metodi richiesti dal progetto e i vari metodi per la lettura e la scrittura negli appositi file.json . Alla creazione di questa istanza di WinsomeServer sarà effettuata l'operazione di "recuperodati" per ricostruire lo stato del sistema. Successivamente il server farà le operazioni per il setup della connessione remota per la RMI (crea lo stub, crea il registry alla porta remota..) , attiverà un thread rewards passandogli come runnable task "RewardsTask",il quale si occuperà di effettuare il calcolo delle ricompense e avvertire i vari utenti collegati del possibile aggiornamento del loro portafoglio ,utilizzando una MulticastSocket, su un indirizzo e porta di multicast(anche il valore di queste si trova nel file di configurazione "server.properties"). In seguito creerà la ServerSocket per accettare le connessioni TCP dei vari client, assegnandole un timeout il cui valore è riportato nel file di configurazione testuale, e istanzierà un CachedThreadPool in modo tale che, ogni qualvolta un client si connetti al server, venga eseguito un nuovo thread del ThreadPool passandogli come runnable task "ClientHendler", il quale si occuperà di gestire tutte le varie operazioni richieste dal client; appunto al "ClientHendler" il server passerà la socket per connettersi al client,l'istanza della classe WinsomeServer creata in precedenza e l'indirizzo e porta di multicast che il ClientHendler comunicherà al client ,in seguito alla login,per potersi connettere e stare in ascolto delle comunicazioni di multicast. Infine, allo scadere del timeout della socket, se nessun altro client si connetterà al server, quest'ultimo aspetterà che i thread del ThreadPool terminino (aspettando nuovamente il tempo del timeout della ServerSocket), farà una shutdown del ThreadPool e si spegnerà. Dunque, come si può evincere, il server è stato realizzato con JAVA I/O e threadpool.

Lato Client:

All'esecuzione del ClientMain.java, verrà eseguito il client che leggerà i vari parametri dal file di configurazione testuale "client.properties"(esso si trova nella cartella "config" dentro la cartella "src"),farà le operazioni per il setup della connessione remota (RMI) e instaurerà una connessione TCP col server. Dopo ciò verrà stampato a schermo un messaggio di benvenuto, il comando per vedere la lista dei comandi disponibili con la corretta sintassi e verrà ricordato che per effettuare le varie operazioni (tranne la registrazione) bisogna prima effettuare il login. La registrazione sarà effettuata tramite RMI richiamando un metodo definito in fondo alla classe. Effettuando la login, se l'operazione è andata a buon fine, il client si iscriverà al servizio di notifica per sapere i nuovi follower (o unfollower)che riceve(tramite RMI Callback),riceverà dal ClientHendler indirizzo e porta di multicast e avvierà un thread "notificawallet" passandogli come runnable task il MultiCastWorker; questo si connetterà all'indirizzo e porta di multicast(i riferimenti dunque vengono forniti dal ClientHendler),riceverà il DatagramPacket che il RewardsTask gli invierà quando avrà effettuato il calcolo delle ricompense con il messaggio di notifica per il possibile aggiornamento del portafoglio e lo stamperà a schermo(il messaggio di notifica sarà "Il tuo wallet potrebbe essere stato aggiornato"). In seguito, si potranno digitare i vari comandi e interagire con il sistema. Infine, effettuando la logout, il ClientHendler associato al client verrà interrotto e terminerà, mentre il client interromperà il thread "notificawallet", si disiscriverà dal servizio di notifica dei follower e terminerà l'esecuzione. È permesso il login simultaneo di più client, questo per dare una simulazione più concreta di un social network. Chiaramente prima di eseguire il client (ClientMain.java) vi è il bisogno si eseguire il server (ServerMain.java).

Threads attivati e controllo della concorrenza

Lato Server:

All'esecuzione del server, verrà attivato il thread rewards, passandogli la runnable task RewardsTask , che prende come parametri d'ingresso l'istanza della classe WinsomeServer(per chiamare il metodo di CalcoloRicompense),indirizzo e porta di multicast(per poter avviare la comunicazione di multicast, il loro valore si trova nel file di configurazione testuale "server.properties" nella cartella "conifg" dentro la cartella "src"), l'intervallo di tempo da aspettare per il calcolo delle ricompense(anche questo valore si trova nel file di configurazione testuale) e la percentuale che spetta all'autore e al curatore/i (anche questi sono nel file di configurazione). In seguito, il server crea un CachedThreadPool e, alla ricezione di connessione di un client, crea un nuovo thread del threadpool per ogni client passandogli come runnable task ClientHendler. Questa, come descritto in precedenza, gestirà tutte le varie richieste; come parametri d'ingresso prende l'istanza di WinsomeServer(che userà per chiamare i vari metodi implementati su esso),la socket per interagire col client tramite StreamBased I/O e l'indirizzo e porta di multicast per poterli passare al client dopo l'avvenuta login affinchè quest'ultimo abbia i riferimenti per mettersi in ascolto sulle comunicazioni di multicast. Ogni metodo implementato in WinsomeServer, che i vari ClientHendler chiameranno per esaudire le varie richieste dei client

(sempre sulla stessa istanza della classe WinsomeServer), è synhcronized per avere una lock implicita sul determinato metodo; questo per garantire mutua esclusione sull'esecuzione dei metodi da parte dei threads. In più ho introdotto delle ReentrantReadWriteLock per la gestione di casi critici in cui ogni metodo di WinsomeServer ha una lock in scrittura o lettura (in base al metodo). Ad esempio, se due ClientHendler diversi richiedono simultaneamente uno l'operazione di delete di un post e un altro di votare quel post, il primo entra nel metodo per la delete e acquisisce la lock in scrittura in modo tale che il secondo anche se entra nel metodo per il rate del post non possa acquisire la lock in scrittura fino a quando il primo non abbia eseguito il metodo e abbia rilasciato la lock.

Lato Client:

All'esecuzione del client (ClientMain.java), dopo aver effettuato il login, il client avvierà un thread "notificawallet" passandogli come runnable task MultiCastWorker; quest'ultimo ha come parametri d'ingresso indirizzo e porta di multicast, i quali ottenuti dal client tramite il ClientHendler associato sulla connessione TCP, e servirà per comunicare al client la notifica di possibile aggiornamento del portafoglio (stamperà a schermo il messaggio di notifica).

Descrizione sintetica delle classi e delle strutture dati usate

ClientHendler:

Runnable task che gestisce le richieste del client (un ClientHendler per ogni client), comunicando con esso tramite la socket (passatagli dal server) e stream based I/O. Come parametri d'ingresso prende la socket, l'istanza di WinsomeServer, l'indirizzo e porta di multicast. Alla ricezione di richiesta di logout da parte del client, si interromperà chiamando la propria interrupt.

ClientMain:

Main per l'esecuzione del client; tramite esso potremmo interagire col sistema da linea di comando (prima però chiaramente avviare ServerMain).

MultiCastWorker:

Runnable task per stare in ascolto sulla comunicazione multicast usando una MulticastSocket. Riceverà il Datagram Packet con il messaggio di notifica del possibile aggiornamento del suo wallet (inviatogli dal RewardsTask) e lo stamperà a schermo (facendolo vedere sulla riga di comando del client). Come parametri d'ingresso prende indirizzo e porta di muticast e verrà avviato dal client dopo aver eseguito il login con successo (il client riceve dal ClientHendler associato i riferimenti). Quando il client riceverà da command line il conmando di logout, manderà un interrupt al MultiCastWorker.

WinsomeClientInterface:

Interfaccia contenente i metodi esportati dal client e che il server usa per la notifica di un nuovo follower o unfollower e per aggiornare all'inizio la map dei follower (l'aggiorna solamente all'inizio quando un client si registra al servizio, in seguito verrà aggiornata tramite il meccanismo di RMI Callback; questo affinchè se un utente si collega riceve dei follower e dopo fa la logout,quando rientra non deve ricevere la notifica dei follower che gia aveva avuto in precedenza). Vi è anche il metodo chiamato dal client per vedere la propria lista dei followers.

NotifyEventImpl:

Implementazione dell'interfaccia WinsomeClientInterface. Usa, come struttura dati, una ConcurrentHashMap che ha come key l'username, come value un ArrayList di stringhe con i nomi dei follower dell'username. Il client creerà un'istanza di questa classe e, dopo aver effettuato il login, si iscriverà al servizio di notifica e avrà la sua struttura locale della mappa dei follower per controllare la sua lista dei followers.

WinsomeServerInterface:

Interfaccia per i metodi di register e unregister al servizio di callback dei client e del metodo per la registrazione

Post:

Classe usata per l'implementazione dei post. Ogni Post ha un Id (Integer, inizialmente settato a 1 ma quando viene chiamato il metodo per la creazione del post gli verrà assegnato un Id univoco,se non lo è già), l'autore(String), il titolo (String), il contenuto (String), una ConcurrentHashMap<String,ArrayList<String>> per i commenti che ha come key l'autore del commento e come value un ArrayList di stringhe per avere la lista dei commenti dell'autore, una ConcurrentHashMap<String,Integer> per i voti che ha come key l'autore del voto e come value un Integer che identifica il voto dato(può essere o 1 o -1), un booleano rewinPosts che mi dice se è un post "rewinato" (true se lo è) e un Integer iterated che mi indica quante volte il Post è stato iterato per il calcolo delle ricompense.

N.B. Dato che questa simulazione di social network fa un calcolo di ricompense su ogni Post, ho pensato che non fosse giusto far guadagnare un utente sul rewin di un Post di un altro utente, in quanto il rewin (a mio modo di vedere) serve esclusivamente per dare visibilità a tale Post. Quindi non permetto di votare o commentare un post "rewinato" e nel calcolo delle ricompense non verrano presi in considerazione i Post con rewinPosts=true.

Utente:

Classe utilizzata per l'implementazione di un utente del social. Ogni utente ha un username (String), una password (String) e una lista di tag (ArrayList di stringhe, massimo 5 tag consentiti)

ServerMain:

Main per l'esecuzione del Server. Sarà attivo per accettare connessioni dei client fino a quando non scatterà la ServerSocketTimeout (se per l'intervallo di tempo del timeout della serversocket nessun client si connette), dopo di che terminerà la sua esecuzione.

RewardsTask:

Runnable task per attuare il calcolo delle ricompense e per mandare il messaggio di possibile aggiornamento del portafoglio ai client connessi. Verrà avviato all'esecuzione del server e prende come parametri d'ingresso l'indirizzo e porta di multicast, l'intervallo di tempo che deve passare per poter svolgere il calcolo delle ricompense, la percentuale di guadagno che spetta all'autore e che spetta al curatore e l'istanza della classe WinsomeServer per chiamare il metodo del calcolo delle ricompense (il valore di tutti questi parametri sta nel file di configurazione testuale server.properties). All'avvio RewardsTask andrà in sleep per l'intervallo di tempo prestabilito, dopo di che effettuerà il calcolo delle ricompense, manderà sulla porta di multicast il messaggio di notifica dei wallet e stamperà a schermo (quindi sulla command line del server) che il calcolo è stato eseguito.

WinsomeServer:

Classe che implementa i metodi di WinsomeServerInterface per la notifica dei followers, tutti i metodi richiesti dalla consegna, il metodo per il calcolo delle ricompense, i metodi per leggere e scrivere dai file.json e per il recupero dei dati (ci saranno prima i metodi di WinsomeServerInterface, poi i metodi della consegna del progetto in ordine di come sono riportati nel pdf della consegna, successivamente ci sarà il metodo per il calcolo delle ricompense e i metodi di lettura e scrittura dei file.json). Come strutture dati utilizza un ArrayList<Utente> per memorizzare gli utenti registrati, una ConcurrentHashMap<String,ArrayList<Post>> per memorizzare i Post(la key sarà l'autore del post, il value sarà la lista di post di cui è autore o dei post di cui ha fatto il rewin), una ConcurrentHashMap<String,ArrayList<String>> per memorizzare per ogni utente(key) i suoi follower (value) e analagomante un'altra per i following. Successivamente usa una ConcurrentHashMap<String,Double> per memorizzare per ogni utente (key) il valore del suo portafoglio (value), una ConcurrentHashMap<String,Map<String,Double>> per memorizza per ogni utente (key) la lista delle transazioni (value è una Map che ha come key la data della transazione e come value l'incremento del wallet), una ConcurrentHashMap<Integer,ArrayList<String>> per tener conto dei voti che ho conteggiato nel calcolo delle ricompense (la key è l'Id del Post e il value è la lista degli utenti che hanno votato positivamente quel determinato Post), e infine una ConcurrentHashMap<Integer,ConcurrentHashMap<String,ArrayList<String>>> per tener conto dei commenti conteggiati nel calcolo delle ricompense (la key è l'Id del Post la value è una map che ha per key l'username dell'utente e value la lista dei commenti conteggiati di quell'utente). Chiaramente ognuna di queste strutture dati verrà scritta nei corrispettivi file.json.

Test:

Nel progetto ho lasciato alcuni test (non tutti) che ho fatto per testare le varie funzionalità del sistema (si possono notare aprendo i file.json). In particolar modo vi sono esempi di utenti registrati, di followers e following che hanno alcuni utenti, un esempio di calcolo delle ricompense di un post avente 3 commenti e 1 voto positivo da parte di un utente,1 commento e 1 voto positivo da parte di un altro, 1 commento e 1 voto positivo da parte di un altro e 1 voto negativo da parte di un altro ancora; consequenzialmente nei file.json dei wallets e transazioni si può

notare il guadagno di ognuno di essi e la transazione aggiunta, mentre nel file.json dei votiConteggiati e commentiConteggiati i vari voti e commenti conteggiati del Post.

Istruzioni per la compilazione del progetto:

Sebbene il progetto sia già compilato ed eseguibile, lascerò qui di seguito le istruzioni per la compilazione (per il progetto ho usato la libreria gson-2.8.9.jar).

N.B: La compilazione ed esecuzione del progetto è stata svolta sulla command line di WINDOWS

Per compilare il progetto bisogna innanzitutto entrare da linea di comando nella cartella del progetto “RETIPROJECT” e successivamente entrare nella cartella “src”, dopo di che digitare il comando:

javac -cp ".;./gson-2.8.9.jar" ClientHendler.java ClientMain.java MultiCastWorker.java NotifyEventImpl.java Post.java RewardsTask.java ServerMain.java Utente.java WinsomeClientInterface.java WinsomeServer.java WinsomeServerInterface.java

Istruzioni per l’esecuzione del progetto:

Per eseguire il progetto bisogna entrare da command line nella cartella “RETIPROJECT”

N.B: È chiaramente necessario eseguire prima il server e poi il client affinchè si possa testare il progetto

Istruzione per eseguire il server:

Una volta dentro la cartella “RETIPROJECT” digitare:

java -cp \".;./bin/;./lib/gson-2.8.9.jar\" ServerMain

Istruzione per eseguire il client:

Una volta dentro la cartella “RETIPROJECT” digitare:

java -cp \".;./bin/;./lib/gson-2.8.9.jar\" ClientMain

Istruzione per eseguire il .jar del server:

Una volta dentro la cartella “RETIPROJECT” digitare:

java -jar Server.jar

Istruzione per eseguire il .jar del client:

Una volta dentro la cartella “RETIPROJECT” digitare:

java -jar Client.jar

Sintassi dei comandi per eseguire le varie operazioni:

Di seguito sono riportati i comandi per eseguire tutte le operazioni richieste con la corretta sintassi (da parte del client):

REGISTRAZIONE (max 5 tag): register username password tag1 tag2 tag3 tag4 tag5

LOGIN: login username password

LOGOUT: logout

VEDERE LISTA FOLLOWERS: list followers

VEDERE LISTA FOLLOWING: list following

VEDERE LISTA USERS: list users

FOLLOWARE UN UTENTE: follow username

UNFOLLOWARE UN UTENTE: unfollow username

CREARE UN POST (le parentesi <> sono obbligatorie): post <titolo> <contenuto>

VEDERE IL PROPRIO BLOG: blog

VEDERE IL PROPRIO FEED: show feed

VEDERE UN POST SPECIFICO: show post IdPost

CANCELLARE UN POST: delete IdPost

FARE IL REWIN DI UN POST: rewin IdPost

VOTARE UN POST (il voto può essere 1 o -1): rate IdPost Voto

COMMENTRE UN POST (le parentesi <> sono obbligatorie): comment <IdPost> <commento>

VEDERE IL PORPRIO PORTAFOGLIO (saranno messe a schermo anche le transazioni): wallet

VEDERE IL PORPRIO PORTAFOGLIO IN BITCOIN: wallet btc

VEDERE LA LISTA DEI COMANDI CON LA CORRETTA SINTASSI: help