UNIVERSITÀ DI PISA

RELAZIONE DEL PROGETTO
“**WORD QUIZZLE**”

RETI DI CALCOLATORI - LABORATORIO

A.A. 2019/2020

Studente
Emanuele Urselli

Docente
Federica Paganelli

Descrizione generale

In questa sezione della relazione sarà fornita una descrizione generale dell'architettura del sistema e inoltre, saranno motivate le scelte di progetto.

Come da specifiche, l'applicazione è implementata secondo un'architettura client-server.

Client

Un requisito per avviare correttamente il client, è quello di avviare prima il server.
Nel caso il client fosse avviato prima del server, verrebbe subito terminato dopo aver mostrato il messaggio relativo alla terminazione.
Nel caso opposto, subito dopo il suo avvio, il client mostrerà il nome dell'applicazione, seguito da '>', per indicare la possibilità per l'utente, di digitare il comando desiderato.
Prima di eseguire i vari comandi, è necessario loggarsi attraverso il comando 'login'.
Questo non è vero nel caso del comando 'registra_utente', eseguibile solo se non ancora loggati.
Subito dopo essersi loggati, verrà avviato un thread in ascolto per le richieste di sfida.
Lo stesso, verrà ovviamente terminato nel caso di logout.
Nel caso di sfida, il client procederà all'avvio di thread dedicati rispettivamente alla lettura delle parole/report ricevute/o dal server e, alla scrittura delle traduzioni digitate dall'utente.
Se durante l'esecuzione del client, si presentasse un errore o il server terminasse la sua esecuzione, anche il client, dopo aver mostrato il solito messaggio relativo all'accaduto, andrebbe verso la terminazione della sua esecuzione.

Server

Come accennato nel sezione dedicata al client, il server deve essere la prima componente ad essere avviata.
Al suo avvio, il server procederà all'avvio del servizio che permette agli utenti di registrare nuovi utenti.
Subito dopo, utilizzando NIO, sarà pronto per ricevere le varie richieste dai client.
La scelta di utilizzare NIO, è legata a tutti i vantaggi legati a quest'ultima tecnologia, quando implementata in un server web.
Le varie richieste dei client, vengono gestite attraverso il main thread.
Differente la situazione nel caso di sfida: alla ricezione del comando 'sfida' da parte di un client (lo sfidante), sarà avviato un thread che provvederà all'inizializzazione della sfida, ovvero, all'invio della richiesta di sfida al client sfidato.
Se la risposta dello sfidato è positiva, il server avvierà un altro thread che gestirà completamente la sfida.

Il thread gestore della sfida, utilizza un proprio selettore, al quale sono registrate le chiavi relative ai player, dopo aver azzerato il loro interest set nel main selector.
Al termine della sfida, gli interest set delle rispettive chiavi saranno resettati.
Se durante l'esecuzione, un client termina improvvisamente la sua esecuzione, il server procede alla disconnessione dell'utente loggato attraverso il suddetto client.
In alcuni casi, come IOException sollevata nel tentativo di chiudere una socket, il server verrà terminato.

Ulteriori dettagli, relativi sia al client che al server, saranno descritti e trattati nelle sezioni dedicate.

Scelte progettuali

1. Se nella chiusura di socket, viene sollevata IOException, l'esecuzione della componente, che sia server o client, viene terminata.
2. Le eccezioni lato client, provocano la sua terminazione.
3. Il server è single thread ed utilizza NIO. La sfida è gestita da un thread che al suo interno utilizza NIO, al quale registra i channel dei player, dopo aver azzerato gli interest set nel main selector del thread.
4. Per gestire la sfida lato client, vengono avviati due thread, uno per la lettura e uno per la scrittura, in modo tale da essere sempre pronti a leggere il messaggio di report.

Schema dei threads attivati

In questa sezione sarà fornito uno schema dei thread attivati da ogni componente.

Client

Richieste di sfida (SfidaUDPClient)

Il client, subito dopo essersi loggato correttamente, provvederà all'avvio di un thread che resterà in ascolto di richieste di sfida dirette all'utente associato al client che ha avviato il thread.
Le richieste di sfida, come da specifiche, sono comunicate attraverso la comunicazione UDP.
Per questo, il thread usufruirà di una DatagramSocket in ascolto sullo stesso numero di porta utilizzato dal socket TCP per le comunicazioni con il server.
La richiesta di sfida, ricevuta dal server, in realtà consiste solo nel nickUtente dello sfidante.
Alla ricezione, dal contenuto del byteBuffer utilizzato nella lettura, verrà dichiarata e inizializzata una stringa contenente la suddetta richiesta.
Questa operazione è effettuata per:

- stampare a video la richiesta di sfida;
- settare il contenuto della struttura dati che indica la presenza di una richiesta di sfida. (Subito dopo questa operazione, viene avviato un ulteriore thread. La sua descrizione verrà trattata nella prossima sezione dedicata)

Questo thread è attivo fino a quando l'utente non effettua il logout.

Reset richiesta di sfida (ThreadEliminaSfida)

Dopo aver settato il contenuto della struttura dati che indica la presenza di una richiesta di sfida, lo stesso metodo chiamato per settare quest'ultima, avvierà un thread che dopo aver atteso un periodo di tempo definito nella classe ParametriCondivisi (descritta in seguito), verificherà la presenza di una richiesta di sfida e, in caso di esito positivo controllerà che questa, sia la stessa richiesta presente al momento dell'avvio del thread. Se anche questa verifica risulta positiva, il thread provvede al reset della richiesta di sfida, utilizzando i metodi della classe SfidaRicevuta (descritta in seguito).
Dopo terminerà la sua esecuzione.

Sfida

In caso di sfida, vengono avviati due thread, utilizzati rispettivamente per la lettura dei messaggi (parole da tradurre o report) provenienti dal server e, la scrittura delle traduzioni da parte dell'utente.

Lettura parole/report dal server (LeggiSfida)

Questo thread viene avviato dal metodo sfida.
Al suo avvio, dichiara e inizializza un byteBuffer che utilizzerà per ricevere messaggi (parole da tradurre o report) dal server.
La lettura è effettuata in un ciclo infinito.
Se viene sollevata una IOException, l'intero client terminerà la sua esecuzione.
In alcuni casi, una lettura che non va buon fine, non solleva la IOException, ma legge -1 caratteri, ergo, è stato inserito un controllo anche per questa situazione.
In questo caso, le operazioni effettuate, sono identiche a quelle eseguite in caso di IOException, ovvero, terminazione dell'esecuzione del client.
Dal contenuto del byteBuffer viene creata una stringa.
Se la stringa contiene 'REPORT', attraverso un booleano presente nella classe FineSfida, si indicherà la fine della sfida e dopo aver stampato un messaggio che richiede all'utente di premere invio per tornare al prompt dei comandi, il thread corrente terminerà la sua esecuzione.
Altrimenti, il contenuto indica una parola da tradurre utilizzando il thread ScriviSfida.
In entrambi i casi, la stringa viene stampata a video.
La terminazione di questo thread è garantita dal fatto che in una partita, ci sarà sempre un messaggio di REPORT.

Scrittura delle traduzioni (ScriviSfida)

Questo thread viene avviato dal metodo sfida.
Al suo avvio dichiara e inizializza un byteBuffer che utilizzerà per la scrittura verso il server.
Viene avviato un ciclo for, che eseguirà K (Inizializzato dal valore presente nella classe ParametriCondivisi) iterazioni.
All'interno di questo ciclo, viene eseguito un ulteriore ciclo infinito in cui verrà presa da stdin la traduzione della parola letta dal thread LeggiSfida.
Subito dopo aver digitato la traduzione, il client verificherà la fine della sfida.
Se l'esito risulta positivo, il valore booleano di fineSfida viene resettato e il thread termina la sua esecuzione.
Se l'esito risulta negativo, si controlla che la traduzione digitata non sia nulla.
Se nulla, il ciclo interno infinito, passerà alla prossima iterazione, chiedendo di digitare di nuovo la traduzione relativa alla stessa parola, altrimenti, si uscirà dal ciclo interno.
Usciti dal ciclo interno, si invierà la traduzione al server.
Se nella scrittura, viene sollevata IOException, l'intero client viene terminato, altrimenti, si passerà alla prossima iterazione del ciclo for.
Dopo aver eseguito le K iterazioni del ciclo for, il thread stamperà a video un messaggio che indicherà all'utente di attendere che l'avversario finisca.
In un altro ciclo controllerà il valore di fineSfida.
Appena settato, lo resetterà e terminerà la sua esecuzione.

Server

Il server per ogni richiesta di sfida, avvia un thread per l'inizializzazione della stessa.
Se l'inizializzazione avviene correttamente, provvede all'avvio di un thread che la gestisca.

Inizializzazione (InizializzazioneSfida)

Questo thread viene avviato dal main thread del server nel momento in cui viene ricevuta una richiesta di sfida da un utente.
Al suo avvio, resetta l'interest set delle chiavi relative ai player impegnati nel main selector del server.
Subito dopo, attraverso il metodo inviaRichiesta, invia la richiesta di sfida all'utente sfidato, utilizzando la comunicazione UDP.
L'indirizzo utilizzato è lo stesso del socketChannel utilizzato per la comunicazione TCP col client, così come per il numero di porta.
In caso di errore, l'interest set delle chiavi del main selector del server, viene resettato a OP_READ e il selector viene risvegliato.
In seguito, il thread di inizializzazione sfida viene terminato.
Altrimenti, dopo aver inviato la richiesta, si attende e si elabora la risposta attraverso il metodo elaboraRisposta.
In questo metodo si attende la risposta per T1 millisecondi (inizializzato in ParametriCondivisi).
Se passati i T1 millisecondi, la risposta non è stata ricevuta, gli interest set delle chiavi vengono resettati e il main selector del server viene risvegliato.
Altrimenti, se si riceve risposta, se negativa vengono effettuare le stesse operazioni riguardanti il main selector del server e le chiavi dei player.
Se positiva, viene avviato un thread per la gestione della sfida vera e propria.
In ogni caso, lo sfidante viene avvisato.
Se si verificano errori durante queste fasi, alcuni possono risultare fatali per l'intero server (come l'IOException sollevata nel caso di socket.close()), mentre altri porterebbero alla disconnessione dell'utente con 'problematiche', il reset degli interest set delle chiavi del main selector nel server e il wakeup dello stesso.
In caso di questi ultimi errori (non fatali), i player impegnati nella inizializzazione della sfida, vengono eliminati da una struttura dati che contiene una lista degli utenti impegnati in una sfida e, in cui erano stati precedentemente inseriti.

Gestore sfida (GestoreSfidaServer)

Questo thread è avviato nel caso l'inizializzazione della sfida vada a buon fine.
Al suo avvio, dopo aver inizializzato i vari parametri, utili allo svolgimento della sfida, chiama il metodo inizializza.
Attraverso questo metodo, inserisce in una struttura dati creata appositamente per mantenere i dati relativi ai punteggi dei due player durante la partita, una voce per ogni giocatore, inizializzando i dati.

Dopo, attraverso il metodo scegliParole, inserisce in una lista di stringhe, tutte le linee di un file contenente le parole che possono essere inviate ai client durante una partita.
Da queste parole, attraverso determinate operazioni, vengono scelte in modo casuale solo K parole che verranno poi, effettivamente utilizzate durante la partita.
Per ogni parola selezionata, viene ottenuta la sua traduzione, interrogando il servizio indicato nelle specifiche di progetto, attraverso il metodo traduciParola..
Alla fine, si avrà una hashmap con le associazioni Parola-Traduzione.
Il metodo scegli parole, ritorna la lista di parole selezionate.
Subito dopo, viene creato un selettore per la sfida, in cui vengono registrate le varie chiavi.
Viene avviato un ciclo infinito, in cui si controlla se il tempo passato sia maggiore o uguale a quello definito in ParametriCondivisi oppure se i client che hanno finito siano entrambi.
Se nessuna delle due condizioni è vera, si controlla se almeno un client abbia avuto 'problemi' di comunicazione' e nel caso, gli utenti vengono eliminati dalla struttura dati che li indicava come impegnati in una sfida e, il thread termina la sua esecuzione.
Altrimenti, si procede alla gestione delle operazioni di input/output.
Se almeno una delle due condizioni si avvera, l'interest set delle chiavi, all'interno del selettore della sfida, viene settata a OP_WRITE.
Per ogni player, attraverso il metodo calcolaReportEAggiornaPunteggio, viene calcolato il punteggio partendo dai dati temporanei della partita e poi viene restituita una stringa contenente il report, calcolato sulla base dei dati appena menzionati.
Si provvede all'invio del report ai due player e se tutto è andato a buon fine, i player vengono eliminati dalla struttura dati degli utenti impegnati.
Si resettano gli interest set delle chiavi degli utenti all'interno del main selector del server e si risveglia, terminando l'esecuzione del thread gestore della sfida.
Anche in questo caso, ci sono alcuni errori che possono essere fatali per l'intero server e altri che porterebbero al reset delle chiavi all'interno del main selector del server, la chiusura del socket con problemi di comunicazioni, la disconnessione dell'utente e il risveglio del main selector.
Ovviamente, nel caso di disconnessione improvvisa di un player, l'altro verrebbe immediatamente avvisato con un messaggio di report.
Le operazioni di input/output vengono effettuate attraverso i metodi read e write.

- read: dalla selection key ottengo il channel del client. Leggo la traduzione inviata dal client, la comparo con la traduzione restituita dal servizio web e se corretta, incremento il punteggio corrente del player e il numero delle risposte corrette, altrimenti decremento il punteggio corrente e incremento il numero di risposte sbagliate. Viene incrementato il numero all'interno dell'attachment e l'interest set della key viene settato a OP_WRITE. Se si verifica un errore di comunicazione con il client, viene chiusa la socket col client, viene disconnesso, i player vengono eliminati dalla struttura dati degli impegnati e l'altro player viene avvisato con un messaggio di report.
- write: dalla key ottengo il socket channel del client. Se l'intero nell'attachment è uguale a K, viene incrementato il numero di player che hanno finito di tradurre le parole e l'interest set viene azzerato, altrimenti, si ottiene la nuova parola da inviare e la si comunica al client. Anche in questo caso, se si verifica un errore nella comunicazione col client, viene disconnesso, la socket viene chiusa, i player

vengono eliminati dalla struttura degli impegnati e l’altro player viene avvisato con un messaggio di report.

Se si solleva IOException durante la chiusura di un socket, il server termina la sua esecuzione.

Strutture dati

In questa sezione saranno descritte le strutture dati utilizzate.

Client

SfidaRicevuta

Questa classe modella una struttura dati utile alla memorizzazione delle richieste di sfida.
Al suo interno è presente una classe statica denominata Sfida, contenente una Stringa che conterrà il nome dello sfidante e un intero che indicherà il numero della sfida ricevuta.
Insieme, questi due valori, saranno utili alla gestione del reset.
Quando viene chiamato il costruttore di questa classe, ovvero, quando viene effettuato il login, i due valori della classe statica vengono inizializzati rispettivamente a “” e -1.
Quando si riceve una sfida, come descritto prima nella sezione dei thread, viene chiamato il metodo setSfida, a cui viene passato il nome dello sfidante.
Il nome dello sfidante verrà usato per inizializzare il valore della stringa nella classe statica e, verrà incrementato il valore del numero della sfida.
Subito dopo sarà avviato un thread che dopo aver atteso un numero di millisecondi stabilito nella classe ParametriCondivisi, verificherà la presenza di una sfida.
Se la sfida è presente, si procederà a verificare se il nome dello sfidante e il numero della sfida sono gli stessi di quelli al momento dell’avvio del thread.
In caso positivo, si provvederà al reset della sfida.
Infine, il thread terminerà la sua esecuzione.
I controlli appena descritti vengono effettuati attraverso un metodo denominato checkSfidaUguale e il reset attraverso un metodo denominato resetSfida.

Server

Il server oltre alle strutture dati che saranno descritte di seguito, utilizza anche una hashmap che mantiene un’associazione tra nickutente e SelectionKey, inizializzata ad ogni avvio del server e ‘popolata’ dai soli utenti connessi.

Grafo

Il grafo è la struttura dati utilizzata per memorizzare i dati di tutti gli utenti di Word Quizzle.
Per il grafo si utilizza il pattern singleton.
La struttura dati è implementata come una concurrentHashMap<String, Utente>.

Chiamando il costruttore, si proverà a deserializzare il contenuto del file al pathname definito nella classe ParametriCondivisi, altrimenti, se il file è vuoto, si inizializzerà la concurrentHashMap.
Attraverso il metodo ottieniGrafo è possibile ottenere, in modo concorrente, il riferimento alla struttura dati.
Attraverso diversi metodi, è possibile andare a modificare il contenuto della struttura dati.
Importante sottolineare che, per ogni modifica della struttura dati, l'intero grafo è riscritto su file.
Attraverso il metodo aggiungiUtente, è possibile aggiungere un nodo al grafo, dopo aver superato gli adeguati controlli.
Attraverso il metodo esisteUtente, è possibile controllare se all'interno della struttura dati esista un utente con il nickUtente passato.
Attraverso il metodo aggiungiAmico è possibile aggiungere alla lista di amici di nickUtente passato, l'utente definito nickAmico.
Attraverso il metodo esisteRelazione è possibile controllare se esista già una relazione di amicizia tra i due utenti definiti dai due nick passati come argomenti.
Attraverso il metodo ottieniClassifica è possibile ottenere la classifica degli amici dell'utente definito dal nickutente passato come argomento.
Per ordinare la lista di amici, viene utilizzato il metodo ordinaHashMap.
Infine, attraverso il metodo aggiornaPunteggio, è possibile aggiornare il punteggio corrente di un utente definito dal socketChannel passato come argomento.
I metodi della classe Grafo vengono chiamati in seguito alla ricezione dei comandi inviati dai client, quindi, possono essere chiamati solo da utenti online.
Per questo, come nel caso del metodo aggiornaPunteggio, usufruisce di un'altra struttura dati: StrutturaLogin, che mantiene un riferimento di tutti gli utenti attualmente connessi.

StrutturaLogin

StrutturaLogin è la classe che modella la struttura dati che mantiene un riferimento di tutti gli utenti attualmente online.
Anche questa struttura dati è implementata con il pattern singleton.
Utilizza una hashmap<SocketChannel, String>, ovvero, un'associazione tra socketChannel e nickUtente.
Attraverso il metodo ottieniStrutturaLogin è possibile ottenere il riferimento, in modo concorrente, alla struttura dati.
Questa struttura dati è inizializzata ogni volta che il server viene avviato.
Attraverso il metodo ottieniSocketChannel è possibile ottenere il socketChannel a partire dal nickUtente e viceversa con il metodo ottieniUtente.
Attraverso il metodo connettiUtente è possibile indicare l'utente definito dal socketChannel passato come argomento, come connesso, aggiungendolo alla struttura dati.
Invece, attraverso il metodo logout è possibile eliminarlo dalla struttura dati.
I restanti metodi, fondamentali come utenteConnesso, sono comunque di appoggio per quelli appena descritti.

Impegnati

Impegnati è la classe che modella la struttura dati che mantiene un riferimento di tutte le coppie di utenti impegnate.
Per impegnate, si intende impegnate in una richiesta di sfida oppure in una sfida vera e propria.
Utilizzando questa struttura dati è possibile evitare che vengano sfidati utenti già in game.
Anche per questa struttura dati è stato utilizzato il pattern singleton ed è implementata come una lista di stringhe.
Come per le precedenti strutture dati, anche questa è inizializzata ogni volta che viene avviato il server.
Attraverso il metodo ottieniImpegnati è possibile ottenere un riferimento, in modo concorrente, alla struttura dati.
La classe prevede solo due metodi.
Il metodo aggiungiCoppia permette di ‘impegnare’ la coppia di player e attraverso il metodo eliminaCoppia è possibile effettuare esattamente l’opposto.

Descrizione delle classi

In questa sezione, sarà fornita una descrizione delle classi definite e delle indicazioni precise sulle modalità di esecuzione.
Alcune classi (Grafo, Impegnati, SfidaRicevuta, StrutturaLogin, InizializzazioneSfida, GestoreSfidaServer, SfidaUDPClient) sono state descritte dettagliatamente nelle sezioni precedenti, quindi, qui saranno trattate quelle non ancora considerate.

ParametriCondivisi

ParametriCondivisi è una classe comune a client e server, contenente tutte le stringhe che definiscono i comandi, i messaggi scambiati tra client e server e i vari parametri per gestire le sfide.

Client

MainClassWQClient

MainClassWQClient identifica il client di word quizzle, attraverso il quale inviare comandi al server.
È possibile passare come argomenti:

- --help: Passando questo argomento, si otterrà un messaggio di USAGE del client ma poi, l'esecuzione verrà terminata.
- indirizzo del server
- porta del server

Questi argomenti, se presenti, serviranno ad inizializzare i valori che poi saranno utilizzati nella connessione con il server.
Se non settati, i valori resteranno quelli di default.
Subito dopo, viene effettuata la connessione con il server, viene inizializzata la datagramSocket per la ricezione delle richieste di sfida via UDP, utilizzando lo stesso numero di porta utilizzato per la comunicazione TCP.
Viene inizializzato l'oggetto scanner per gli input da stdin, utilizzato anche nella sfida e poi viene avviato un ciclo infinito, all'interno è possibile digitare comandi ed elaborarli.
Come già accennato, l'operazione registra_utente è possibile farla se non si è già loggati, mentre le altre richiedono il requisito contrario.
Quindi, come da specifiche, il login è l'operazione da effettuare prima di poter eseguire le altre.
Dopo aver digitato un comando, quest'ultimo viene passato come argomento al metodo elaboraComando.
elaboraComando tokenizza il comando e se il primo token è:

- registra_utente: controlla se si è loggati e nel caso stampa un messaggio di errore, altrimenti, chiama il metodo registrazione passando il valore dei seguenti due token che corrispondono a nomeUtente e password. Il metodo registrazione, invoca il metodo RMI

del server che andrà ad aggiungere un nuovo nodo al grafo, con il quale, successivamente, dopo previo login, sarà possibile eseguire le operazioni standard.
- login: controlla se si è loggati e nel caso stampa un messaggio di errore, altrimenti, ottiene il nickUtente dal secondo token, chiama il metodo inviaComando, attraverso il quale invia il comando al server e memorizza la risposta in una stringa. Dopo averla stampata controlla che questa sia positiva e nel caso, avvia un thread per ascoltare le richieste di sfida e setta dei parametri con i dati dell'utente con il quale ci si è loggati.
- logout: controlla se si è loggati e nel caso, ottiene la risposta attraverso invioComando, resetta le sfideRicevute, ferma il thread in ascolto per le richieste e resetta i parametri inizializzati con il login.
- mostra_punteggio: si limita a stampare la risposta ottenuta attraverso invioComando.
- lista_amici: attraverso invioComando, ottiene una risposta in formato JSON, la deserializza in una lista di stringhe e la stampa.
- mostra_classifica: esegue le stesse operazioni del comando appena descritto, ma la risposta la deserializza in una linked hash map e stampa il contenuto.
- aggiungi_amico: come mostra_punteggio si limita a stampare la risposta ottenuta attraverso invioComando.
- sfida: dal prossimo token ottengo il nome dell'avversario e invio il comando. Stampo la risposta ottenuta e se positiva chiamo il metodo sfida.

Se l'esito della sfida non è positivo, come per tutti gli altri metodi, dopo aver eseguito le operazioni descritte, si passa alla prossima iterazione, ovvero, al prossimo comando.
Chiamando il metodo sfida, si stampa un messaggio di inizio sfida, e si avviano i thread LeggiSfida e ScriviSfida descritti prima. Dopo averli avviati, se ne attende la terminazione.
Alla fine della sfida, la sfidaRicevuta viene resettata.
Il metodo inviaComando, prende in input il comando e lo invia al server.
Subito dopo attende la risposta del server e la restituisce al chiamante.
Se si verifica un errore di comunicazione col server, il client viene terminato.

Server

ServerWQ

La classe ServerWQ implementa la componente server di WordQuizzle.
Al suo avvio, avvia il servizio di registrazioneWQ, che attraverso RMI permette di registrare nuovi utenti.
Subito dopo, con gli argomenti da linea di comando, inizializza i parametri che saranno utilizzati per la server socket.
Se non presenti, vengono utilizzati quelli standard.
Infine, viene chiamato il metodo avviaSistema: attraverso questo metodo viene aperto il main selector del server, viene inizializzato il thread pool per l'inizializzazione delle sfide e la struttura dati che associa i nickUtenti alle selection key.
Dopo inizia a gestire le operazioni di input e output, continuando ad accettare eventuali nuove connessioni.
Le connessioni vengono accettate con il metodo accept, che registra il socketchannel del client al main selector dopo averlo configurato in modalità non bloccante.

La lettura avviene attraverso il metodo read, al quale viene passata la selection key corrente.
Dalla selection key, ci si ricava il channel, dal quale si legge il comando inviato dal client.
Se si verificano errori di comunicazione col client, quest'ultimo viene disconnesso e cancellato dal main selector.
Se si solleva una IOException nella chiusura del socket, il server termina la sua esecuzione.
Altrimenti, il comando ricevuto viene elaborato dal metodo elabora.
Il metodo elabora, tokenizza il comando e dalla key, ottiene il channel.
Se il comando è:

- sfida: ottiene il nickUtente, dal secondo token ottiene il nickUtente dell'avversario. Dopo aver controllato che esista una relazione tra lo sfidante e lo sfidato e che lo sfidato sia connesso, se non ci sono problemi, viene inizializzata la sfida attraverso un thread descritto prima. Altrimenti, viene generata la risposta relativa all'errore e inviata allo sfidante.
- login: ottiene il nickUtente dal secondo token e attraverso i metodi della classe che modella la struttura dati dei loggati, lo aggiunge alla hashmap e genera la risposta.
- logout: ottiene il nickUtente dalla struttura login, lo disconnette dalla stessa, rimuove l'occorrenza dalla hashmap con le associazioni SelectionKey - nickUtente.
- mostra_punteggio: ottiene il nickUtente dalla strutturaLogin e dal riferimento all'oggetto utente, ottiene il punteggio utente e genera la risposta.
- lista_amici: ottiene il nickUtente dalla strutturaLogin e dal riferimento all'oggetto utente, ottiene la lista di amici e genera la risposta.
- mostra_classifica: ottiene il nickUtente dalla strutturaLogin, chiama il metodo ottieniClassifica della classe Grafo e genera la risposta.
- aggiungi_amico: come gli altri comandi, ottiene il nickUtente dalla strutturaLogin e il nickAmico dall'altro token. Dopo, chiama il metodo aggiungiAmico della classeGrafo e genera la risposta;

La risposta generata viene inserita come attachment alla selection key e poi viene settato l'interest set della stessa a OP_WRITE.
La scrittura invece, avviene attraverso il metodo write.
Dalla key si ottiene il channel e si scrive al client il contenuto dell'attachment della selection key.
Infine si risetta l'interest set della stessa key a OP_READ.
Nel caso di errore, si chiude il channel, si disconnette l'utente e se nella chiusura viene sollevata IOException, il server termina la sua esecuzione.

Utente

La classe Utente modella l'utente.
Contiene la password, il punteggio utente e la lista di amici.
Chiamando il costruttore, la password viene inizializzata con quella passata, il punteggio viene settato a zero e la lista di amici è inizializzata come un arraylist di stringhe.
Attraverso il metodo:

- ottieniPassword: è possibile ottenere la password. Questo metodo è utile per il login.
- aggiornaPunteggio: passando il nuovo punteggio, è possibile aggiornare quello attuale.

- ottieniPunteggioUtente: con questo metodo è possibile ottenere il punteggio utente.
- ottieniListaAmici: con questo metodo è possibile ottenere la lista di amici
- aggiungiAmico: con questo metodo è possibile aggiungere un amico

Il metodo toString è utilizzato all'interno della serializzazione.

ImplementazioneRegistrazioneWQ

Questa classe offre il metodo RMI per la registrazione di nuovi utenti.
L'implementazione, implementa l'interfaccia RegistrazioneWQ..
Il metodo, denominato registra_utente, ottiene il riferimento del grafo, in modo concorrente e chiama il metodo aggiungi_utente che restituisce un codice: se 10, il nickUtente è stato già utilizzato, altrimenti, la registrazione è avvenuta correttamente.

Istruzioni per la compilazione e l'esecuzione

In questa sezione, saranno indicate le istruzioni per compilare ed eseguire correttamente il progetto.

Istruzioni per la compilazione da linea di comando

javac -cp .:gson-2.8.2.jar ServerWQ.java
javac -cp .:gson-2.8.2.jar MainClassWQClient.java

Istruzioni per l'esecuzione da linea di comando

java -cp .:gson-2.8.2.jar ServerWQ
java -cp .:gson-2.8.2.jar MainClassWQClient

Sintassi dei comandi

- Registra utente: registra_utente <nickUtente> <password>
- Login: login <nickUtente> <password>
- Logout: logout
- Aggiungi amico: aggiungi_amico <nickAmico>
- Lista amici: lista_amici
- Mostra classifica: mostra_classifica
- Mostra punteggio: mostra_punteggio
- Sfida: sfida <nickUtente>

Durante la sfida, basta inserire il la traduzione della parola e premere invio.
Alla fine della sfida, premere invio per tornare al prompt dei comandi.

Ad una richiesta di sfida, è necessario rispondere 'si' o 'no'.

Librerie esterne

gson-2.8.2.jar

Note

Il progetto è stato sviluppato con JDK 9.
In ogni caso, non dovrebbero essere stata utilizzata alcuna ‘funzione’ presente nella nona versione del linguaggio.