UNIVERSITÀ DI PISA

Dipartimento di Informatica

Laboratorio di Reti (2019-2020)

WQ
Word Quizzle

Autore:

Alberto Marci

mat. 464221

Indice

Capitolo 1

Introduzione

Il progetto consiste nell'implementazione di Word Quizzle (WQ), ovvero un sistema di sfide di traduzione italiano-inglese tra gli utenti che si sono registrati, ed è stato realizzato con una interfaccia grafica basica, solamente per il client che ne facilita l'utilizzo. Come richiesto dalle specifiche il linguaggio utilizzato è Java, e ho avuto la necessita di utilizzare la libreria Json-simple e la libreria Gson. Questa relazione è volta a spiegare l'architettura generale, nonché le scelte progettuali più importanti.

1.1 Struttura del codice

Il codice sorgente di questo programma è stato diviso in due package:

- Server: implementa il lato server del sistema e contiene le varie classi necessarie al corretto funzionamento del gioco.
- Client: implementa il lato client, fornendo oltre a tutti i metodi necessari per interagire con il server anche una semplice interfaccia grafica che ne rende l'utilizzo più intuitivo.

Capitolo 2

Package Server

2.1 ServerMain

Il paradigma utilizzato è quello Client-Server, in cui in un primo momento viene avviato il server, che eseguirà dei metodi di inizializzazione, che comprendono:

- initServer() –> Inizializza le strutture dati necessarie per il corretto funzionamento del server, crea la directory UTENTI nel caso non esistesse e si occupa di ricreare il database degli utenti e delle amicizie fra utenti. Crea la ServerSocket di benvenuto per accettare le connessioni, avvia un ThreadPool Executor per la gestione dei client e si occupa anche di attivare il servizio RMI per la registrazione al server.

- initServerCycle() –> In questo metodo viene avviato il ciclo infinito del server, il cui scopo è attendere che i client si connettano, per poi creare un task (ClientTask) che viene passato al ThreadPool Executor creato e avviato precedentemente.

Per quanto riguarda l'Executor ho scelto di utilizzare il CachedThreadPool, che crea un pool di thread che crea nuovi thread a seconda della necessità, e riutilizza i thread precedentemente creati qualora disponibili.

Il server utilizza due ConcurrentHashMap per tenere traccia degli utenti che si iscrivono al servizio, e per mantenere traccia delle relazioni di amicizia tra gli utenti.

- ConcurrentHashMap<String, User> userList;

- ConcurrentHashMap<String, ArrayList<String>> friendList;

Anche per le liste delle parole in italiano e tradotte si usano due ConcurrentHashMap<Integer, ArrayList<String>>.

Ho scelto questo particolare tipo di HashMap in quanto è Thread-safe, quindi non ha bisogno di ulteriori meccanismi di sincronizzazione anche se ci saranno molti utenti connessi al server che eseguiranno le varie operazioni.

2.2 User

La classe User rappresenta l'utente (lato server) che si iscrive al server Word Quizzle, ed è costituito da:

- String username, nome univoco dell'utente che si registra al servizio.
- String password, password utilizzata per eseguire il login dopo la registrazione.
- Boolean isOnline, rappresenta lo stato dell'utente, true = utente online, false = utente offline.
- Boolean inSfida, serve a capire se l'utente è già impegnato in una sfida con qualche altro utente.
- Int punteggioTotale, punteggio accumulato durante tutte le sfide disputate.
- Int punteggioPartita, punteggio dell'ultima partita effettuata.

Questa classe contiene inoltre tutti i metodi get e set per ottenere o modificare i propri parametri.

2.3 ClientTask

Classe che implementa Runnable e che si occupa della gestione delle richieste da parte del proprio client. Ogni volta che viene accettata una connessione, sul server viene generato un task (ClientTask) che viene passato al Thread Pool Executor. Questo una volta avviato rimarrà

in attesa di ricevere comandi da parte del client, e una volta decodificati ed eseguiti, invierà i risultati delle operazioni eseguite in messaggi di risposta (int o String a seconda del comando richiesto). Quando il client deciderà di chiudere la finestra della GUI avverrà la disconnessione dal server e terminerà anche il ciclo del task. Il client e il ClientTask rimarranno infatti collegati da una connessione persistente durante tutta la durata della vita della GUI. Le operazioni che il ClientTask è in grado di eseguire sono tutte quelle descritte nelle specifiche del progetto.

2.4 GameThread

Classe che implementa Thread e si occupa di inviare al client le parole da tradurre, di ricevere le traduzioni e controllare se la risposta ricevuta corrisponde a quella ottenuta tramite la GET HTTP in modo da aggiornare il punteggio dell'utente di conseguenza.

2.5 Interfacce Remote

Il metodo remoto che realizza la funzionalità di registrazione di un utente al server è definito nell'interfaccia ServerInterfaceRMI. Viene implementato dalla classe ServerImplementationRMI, e oltre a offrire l'operazione di registrazione si occupa anche di modificare il file ListaUtenti.json per la persistenza dei dati.

2.6 ServerConfig

Classe in cui sono memorizzate alcune porte, e alcuni parametri per il corretto funzionamento del server.

Capitolo 3

Package Client

3.1 ClientMain

Classe che contiene il main per avviare la GUI.

3.2 ReceiverUDP

ReceiverUDP è una classe che estende Thread ed implementa un ciclo in cui si resta in attesa di ricevere pacchetti UDP che conterranno come messaggio il nome dell'utente che ha lanciato la sfida. Questo Thread viene attivato nel momento in cui si effettua il login e viene terminato con il logout o con la chiusura della GUI.

3.3 ClientUI

ClientUI Classe che estende JFrame, e che inizializza tutti i componenti della GUI, le strutture dati necessarie per la comunicazione Client-Server, e vengono implementati tutti i metodi per l'invio dei vari comandi al server.

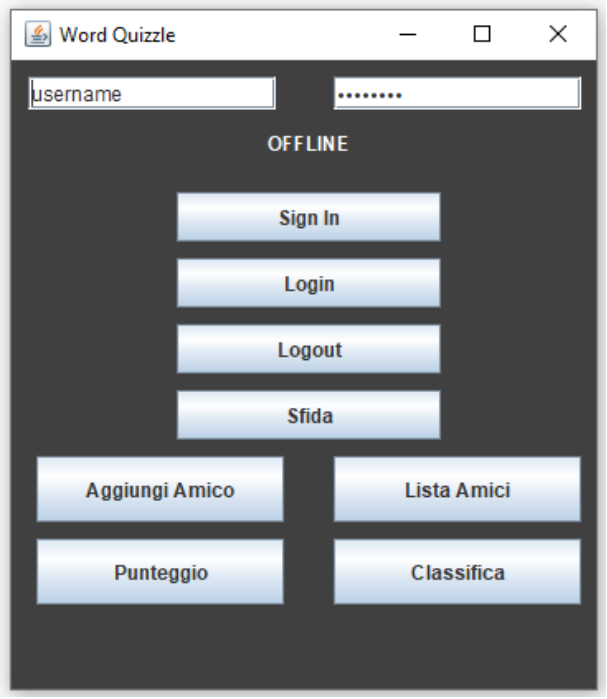

Figura 3.1: **GUI**

3.4 ClientConfig

Classe in cui sono memorizzate alcune porte, e alcuni parametri per il corretto funzionamento del client.

Capitolo 4

Implementazione

4.1 Registrazione

La gestione della registrazione è stata implementata, come richiesto da specifiche con RMI, in modo da invocare il metodo del server in maniera remota, creando l'oggetto remoto e passandolo come stub al Registry: in questo modo lo stub permette al client di usare i metodi definiti remoti lato server. Al momento della registrazione l'utente viene aggiunto sia al file ListaUtenti.json che e al file Amiciczie.json in modo da avere la persistenza dei dati. Per eseguire queste operazioni ho usato la libreria Json-Simple.

4.2 Login

Viene implementato tramite connessione TCP. Durante la fase di inizializzazione una Socket, su localhost e con porta nota, viene utilizzata ottenendo i relativi DataOutputStream e BufferedReader per l'invio di richieste e la lettura di risposte, da e verso il server. Se la richiesta ha successo l'utente passa nello stato online, e viene attivato il Thread ReceiverUDP in modo da poter ricevere richieste di sfida da parte dei giocatori. Se invece la richiesta non ha successo viene aperto un Dialog con un messaggio di errore.

4.3 Logout

Viene implementato tramite connessione TCP. L'utente passa nello stato offline e il ciclo del Thread ReceiverUDP viene terminato.

4.4 Sfida

Viene implementata in parte tramite connessione TCP e in parte tramite UDP. Lato client viene aperto un Dialog in cui inserire il nome del giocatore da sfidare, che viene inviato tramite TCP al server, il quale effettuerà un "controlloPreSfida", in cui verrà verificato se la sfida sia possibile tra i due giocatori. In caso di esito negativo viene aperto lato client un Dialog che ne indicherà la causa con un messaggio di errore appropriato. Se l'esito del controllo è positivo il server preparerà ed invierà un datagramma all'utente da sfidare, contenente come messaggio il proprio username. Una volta inviato tramite UDP il server rimarrà in attesa di una risposta da parte dell'utente sfidato per T1 millisecondi. Se dovesse scattare il time-out la richiesta viene automaticamente rifiutata. L'esito della richiesta viene comunicato all'utente usando TCP. Se la sfida è stata accettata lato server vengono selezionate le N parole e tramite un metodo che effettua la GET HTTP ne viene richiesta la traduzione. Lato client ci si prepara per l'inizio della sfida con il metodo avviaSfida(int N).

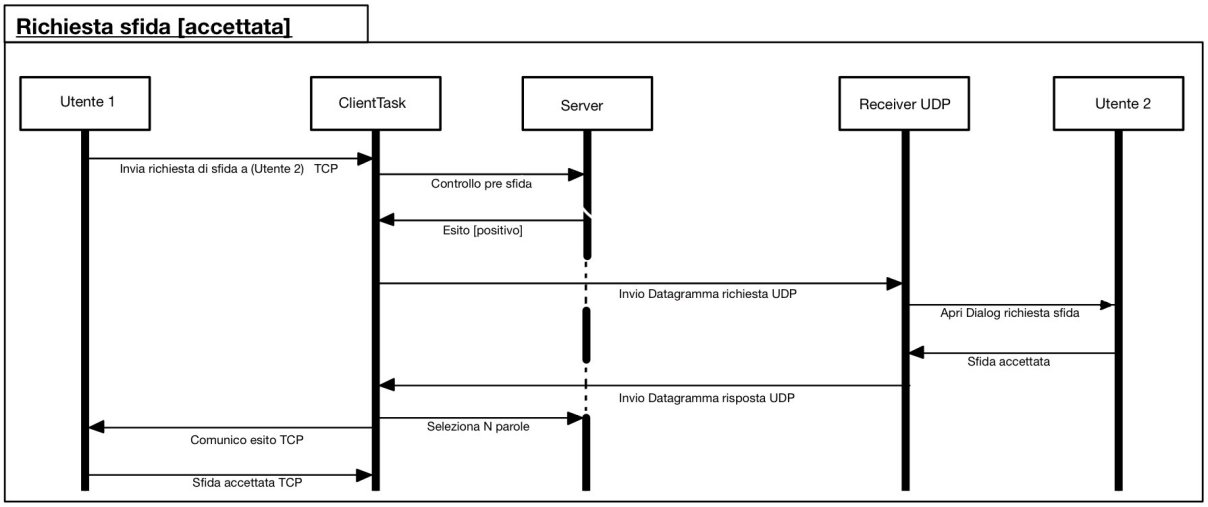

Figura 4.1: **Richiesta di sfida**

A questo punto verrà impostato un timer per T2 millisecondi al termine del quale verrà inviato al gameThread la stringa “INTERRUZIONE” in modo notificargli che il tempo a disposizione è scaduto e interrompere l’invio delle parole restanti. Se le parole da tradurre sono terminate o se è scattato il time-out si prosegue con la fase di aggiornamento dei punteggi e la dichiarazione del vincitore.

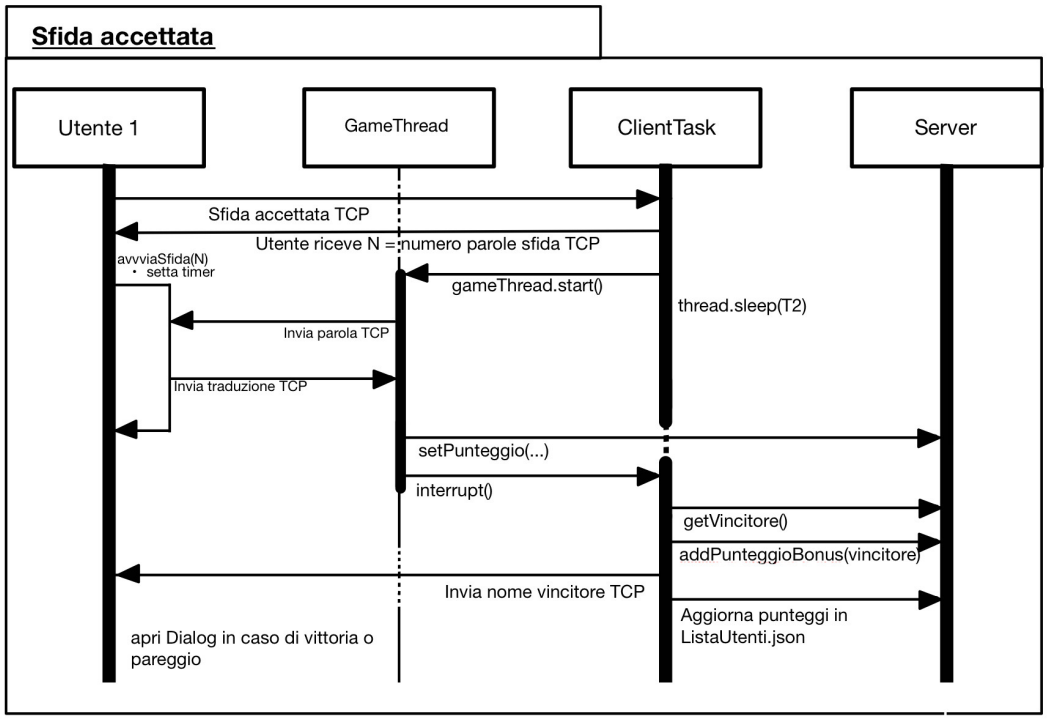

Figura 4.2: **Richiesta accettata**

4.5 Aggiungi Amico

Implementata tramite connessione TCP, viene creato e aperto un Dialog in cui inserire il nome del giocatore da aggiungere alla propria lista delle amicizie. Il nome del giocatore viene inviato al server, che farà alcuni controlli. In caso di esito positivo l’utente verrà aggiunto alla lista e verrà aggiornato il file Amici.json (per la persistenza dei dati). Lato utente un Dialog sarà aperto mostrandoci l’esito della richiesta.

4.6 Lista Amici

Implementata tramite connessione TCP, lato client una volta premuto il JButton lista Amici viene inviato al server il proprio nome e questo restituirà una stringa in formato Json contenente la lista degli amici che verrà mostrata in un Dialog. In caso l'utente non abbia ancora nessun amico, verrà mostrato un apposito messaggio. Ho utilizzato la libreria Gson in questo caso per ottenere una stampa più leggibile.

4.7 Mostra Punteggio

Implementato tramite TCP apre un Dialog e mostra il punteggio totale dell'utente.

4.8 Mostra Classifica

Viene implementato tramite connessione TCP. Nel caso l'utente non abbia nessun amico verrà aperto un Dialog in cui sarà indicato tale risultato. Altrimenti viene richiesta al server la lista delle proprie amicizie, viene creata e ordinata una lista di User e infine viene creata una stringa Json con il formato: {"posizione[i]":{ "username i" : punteggio i}} che verrà spedita al client. Anche in questo caso ho utilizzato la libreria Gson per ottenere una stampa della classifica più leggibile.

Capitolo 5

Test

Per eseguire il test del progetto basta eseguire il main della classe ServerMain, e attendere che il server sia pronto. Dopodiché basterà eseguire anche il main della classe ClientMain ed eseguire la registazione come prima operazione (oppure utilizzare uno dei nomi gia‘ registrati al gioco visualizzabili nel file ListaUtenti.json), tramite il bottone (JButton) Sign In, scegliendo ed inserendo username e password. Un Dialog ci avviserà con l'esito della registrazione. A questo punto potremo eseguire il login, in questo caso un Dialog verrà aperto solo in caso di esito negativo del comando, mentre in caso di esito positivo la statusLabel (JLabel) mostrerà la scritta "ONLINE".

Una volta online, sarà possibile all'utente eseguire tutte le funzionalità richieste nelle specifiche del progetto. Questo progetto è stato testato solo in ambiente Windows utilizzando come IDE IntelliJ IDEA.