Relazione Progetto Laboratorio di Reti a.a.2022/2023
WORDLE: un gioco di parole 3.0

Stefano Ferralis (598594)

Il progetto consiste nella implementazione di WORDLE, un gioco che ha come obiettivo quello di indovinare una parola entro un certo numero di tentativi. È stato realizzato seguendo vari argomenti svolti durante il corso, particolare attenzione è stata dedicata al lato multi threading del Server, mentre per quanto riguarda il lato Social del gioco è stato implementato come un gruppo Multicast dove gli utenti possono interagire tra di loro inviando i risultati al termina di ciascuna partita.
È da precisare che l'invio da parte del Server degli indizi per indovinare la parola Segreta al Client viene fatto, per evitare spoiler, utilizzando:

X Se la lettera inserita non è presenta nella parola segreta.
? Se la lettera inserita è presente nella parola segreta ma non in quella posizione
+ Se la lettera inserita è presente nella parola segreta in quella esatta posizione.

Descrizione generale del Server

Il Server comunica con i Client attraverso una connessione con Socket TCP con l'utilizzo dei Channel e del Selettore di Java NIO. I Comandi inviati dal Client al Server vengono inviati in formato json, usando come Classe "**Comandi**" presente all'interno della cartella "condivisi" la quale raggruppa tutte le classi che verranno usate sia dal Server che dal Client.
Il Thread principale che si occupa della ricezione dei comandi, quando riceve una nuova connessione registra il canale associato e assegna alla key una sessione utente (che verrà successivamente impostata col relativo username dopo che l'utente ha effettuato il login così da permettere di eseguire determinati comandi).
Come detto in precedenza il Thread Principale riceve i comandi, in particolare i comandi *login* e *logout* vengono eseguiti direttamente mentre i restanti comandi vengono trasformati in Task (oggetti Runnable) utilizzando la classe **TaskThreadPool** ed inseriti nella LinkedBlockingQueue del ThreadPoolExecutor per poter essere eseguiti ed a seconda del comando verrà inviato al Client un codice risposta e il relativo risultato. Tutti i metodi che gestiscono la logica del server(Classifica, Lista di Utenti registrati) sono resi Thread-Safe utilizzando i Monitor (metodi synchronized).
La registrazione avviene tramite RMI e l'utente dopo aver effettuato il login si registra alla Callback che gli notifica quando c'è stato un aggiornamento nelle prime 3 posizioni della Classifica consentendogli di aggiornare e stampare la Classifica salvata in locale.
All'avvio il Server legge il file delle parole segrete *words.txt* e periodicamente estrae una parola casuale che determina la partita attuale alla quale gli utenti possono giocare
Un secondo Thread è invece utilizzato per effettuare il logout automatico lato Server degli utenti che risultano disconnessi in modo anomalo, ovvero se non hanno

effettuato il comando di logout prima di disconnettersi, inoltre viene cancellata la loro registrazione alla Callback.
La persistenza è garantita grazie al salvataggio della Classifica e della Lista degli utenti registrati in due diversi file json che mantengono salvato lo stato del Server anche in caso di chiusura anomala in quanto è stato implementato uno Shutdown Hook che salva prontamente i due file json prima della chiusura del Server.

Descrizione generale del Client

Il Client dopo aver letto il file di configurazione, inizializza una classe per gestire la Classifica in modo locale, la quale viene inviata dal Server in seguito alla login e viene tenuta in costante aggiornamento grazie alla Callback.
Viene creato e fatto partire il Thread che si occupa di accedere al gruppo sociale, implementato come gruppo Multicast, e di ricevere messaggi che contengono i tentativi degli utenti di indovinare la parola segreta.
Dopo aver instaurato una connessione TCP con il server, riceve i comandi presi in input da tastiera e ne effettua il parsing (i comandi avranno la seguente struttura: **comando parametro1 parametro2**) ed a seconda del comando ricevuto, invierà o meno il comando al server e riceverà la risposta di avvenuto successo o meno insieme al risultato aspettato.

RMI nel dettaglio

Registrazione

L'utente effettua la registrazione al Server mediante RMI. Il Server all'avvio fa partire il servizio di registrazione andando a creare un registro nel quale esporterà lo stub "WORDLE-REGSERVER:7777". L'interfaccia ***IRegisterService*** implementa il metodo chiamato dal Client per effettuare l'operazione di registrazione lato Server, l'interfaccia viene implementata dalla classe ***RegisterService*** la quale contiene il metodo effettivo per la registrazione. Il Client dunque quando riceve in input il comando register invocherà il metodo che andrà a prelevare un riferimento al registro creato e creerà un oggetto remoto IRegisterService che servirà per poter chiamare la registrazione lato Server.

Callback

La Registrazione/Deregistrazione alla Callback avviene automaticamente dopo aver effettuato il login o il logout. Il Client si registra al servizio che gli consente di essere avvisato al variare delle prime tre posizioni in Classifica, questo è possibile perchè il Server tiene salvati in un apposito struttura dati <Callback, String(Username)> così da poterli prelevare per usarli nell invio della notifica di aggiornamento posizione.
NB: Appena effettuato il login con la relativa iscrizione alla Callback, il Server invia subito la Classifica usando la classe *ClassificaData* (che implementa Serializable) così da non dover aspettare un aggiornamento nelle prime tre posizioni.

La Classifica è tenuta lato Server, viene usata una HashMap<String, Double> (username, punteggio) che viene aggiornata al termine di ogni partita (sia che l'utente vinca o perda), per rendere questa operazione Thread-Safe è stato reso il metodo per l'aggiornamento synchronized. Il punteggio di ciascun giocatore viene calcolato seguendo il Wordle Average Score come da esempio fornitoci, dunque un punteggio più basso indica un utente più bravo.

La Traduzione della parola segreta viene effettuata ad ogni fine partita inviando una richiesta HTTP GET al sito mymemory presente nel testo del Progetto. Seguendo le API del sito, il Server stabilisce una connessione all'Url: https://mymemory.translated.net/api/get?q=**ParolaSegreta**&langpair=en|it, dove "**ParolaSegreta**" è selezionata in base alla partita giocata dall'utente, la risposta viene ricevuta in formato json ed inviata al Client come Stringa.

Comandi

I Comandi implementati seguono tutti la stessa sintassi: **comando(parametri)** inoltre il **comando** è stato reso case insensitive.

- **register username password**
  Effettua la registrazione dell'utente

- **login username password**
  Effettua l'accesso

- **logout username**
  Effettua la disconnessione

- **playWordle**
  Inizia la partita con la parola segreta estratta in quel momento dal Server

- **sendWord parolaIndovinata**
  Invia al Server la “parolaIndovinata” che rappresenta il tentativo dell’utente di indovinare la parola segreta. Viene restituita una Stringa che rappresenta gli indizi che il Server manda al Client per aiutarlo ad indovinare.

- **sendMeStatistics**
  Richiede al Server le statistiche dell’utente aggiornate dopo l’ultima partita, verranno stampate a schermo

- **share**
  Richiede al Server l’invio sul gruppo Multicast dei risultati dell’ultima partita giocata dall’utente.
  NB: l’utente ha tempo fino all’estrazione della successiva parola segreta per poter richiedere l’invio dei risultati, dopodichè dovrà giocare un’altra partita per poter utilizzare il comando.

- **showMeSharing**
  Mostra a schermo tutti i tentativi inviati al gruppo Multicast da tutti gli utenti

- **showMeRanking**
  Mostra a schermo la Classifica salvata in locale.

Compilazione ed Esecuzione

La cartella principale *Wordle* del file zip è composta da due sottocartelle:
src che contiene tutte le classi .Java del progetto raggruppate a seconda della loro funzione nelle cartelle *server*, *client* e *condivisi*.
jars contiene invece la cartella ***lib*** (contenente le librerie esterne usate, solo la versione 2.10 di gson) e i due manifest.txt; successivamente alla compilazione verrà creata un'altra cartella chiamata **artefacts** che conterrà i file .class e i file .jar da poter eseguire.
Il file .zip contiene già tutte le cartelle comprese di tutto il necessario insieme ai file .jar già compilati e pronti per essere eseguiti. Per eseguire il codice scrivere i due comandi presenti in fondo alla relazione.

Per la compilazione all’interno della directory jars basterà scrivere queste righe:

```
md artefacts
javac ../src/condivisi/*.java ../src/condivisi/interfacce/*.java -d ./artefacts -cp ./lib/2.10/gson-2.10.jar
cd ./artefacts
jar cf ./WordleCondivisiLib.jar ./condivisi/*.class ./condivisi/interfacce/*.class
cd..
```

```
javac ../src/client/*.java -d ./artefacts -cp ./artefacts/WordleCondivisiLib.jar;./lib/2.10/gson-2.10.jar
cd artefacts
jar cfm WordleClient.jar ../manifest-client.txt ./client/*.class ./condivisi/*.class
./condivisi/interfacce/*.class
cd..

javac ../src/server/*.java ../src/server/admin/*.java ../src/server/domini/*.java ../src/server/servizi/*.java
-d ./artefacts -cp ./artefacts/WordleCondivisiLib.jar;./lib/2.10/gson-2.10.jar
cd artefacts
jar cfm WordleServer.jar ../manifest-server.txt ./server/*.class ./server/admin/*.class
./server/domini/*.class ./server/servizi/*.class ./condivisi/*.class ./condivisi/interfacce/*.class
```

ed infine all'interno di jars/artefacts (se si utilizza Windows) scrivere:

```
copy "..\lib\2.10\gson-2.10.jar" .
```

Questo permette l'esecuzione dei due file jar principali

Per eseguire il codice basterà andare in jars/artefacts e scrivere
java -jar WordleServer.jar per avviare il Server
java -jar WordleClient.jar per avviare il Client