LABORATORIO RETI DI CALCOLATORI

Progetto WORDLE

“UN GIOCO DI PAROLE 3.0”

# Relazione di Raoul Morucci, Matricola 580102

# Data consegna 12/01/2024

# DESCRIZIONE GENERALE DELL’ IMPLEMENTAZIONE

Wordle è un programma Java formato da due entità principali; un server che denomineremo WordleServer ed un client che denomineremo WordleClient. Queste due entità formano insieme un applicativo con paradigma client-server che prende spunto dal celebre gioco “Wordle” della nota azienda “Bending Spoons”.

Il WordleClient rappresenta la parte applicativa con cui gli utenti possono interagire, ha un carico minimo in termini di operazioni e gestione di esse. Infatti, il WordleClient si occupa di fornire un’interfaccia minimale e dei set di istruzioni che tramite linea di comando permettono di visualizzare dati di gioco e menù di gioco, altre invece permettono di interagire con il WordleClient andando a modificare dati utente e dati di gioco, completano così un’esperienza utente chiara ed intuitiva. La scelta di rendere così semplice e leggera la parte client è derivata dalla consapevolezza di avere il server applicativo che si occupa della parte più massiccia del carico di gestione. Il WordleServer è stato pensato in modo da poter gestire e garantire stabilità a più di un WordleClient, rendendo così Wordle a tutti gli effetti un applicativo multigiocatore. Il server riesce a servire ed ascoltare ogni client basandosi sul concetto del multiplexing, per implementarlo viene usato un Selector che associa ad ogni client un canale input/output dedicato (JNIO) o anche chiamato SocketChannel, il monitoraggio dei canali, quindi, viene eseguito in modo non bloccante (controlla quando un canale è pronto in I\O ma senza aspettare). Nel seguente caso NIO svolge un ruolo fondamentale nella gestione dei canali, permettendo di leggere/scrivere solo su canali su cui il client ha già inviato/aspetta dei dati. All’ opposto il client utilizza un tipo di monitoraggio bloccante sulla socket (JIO), perché il protocollo di comunicazione utilizzato prevede che il client dopo aver inviato dei byte si metta in ascolto aspettando una risposta dal server. Per parallelizzare la gestione dei task lato server viene implementato il multithreading tramite una FixedThreadpool che varia in base al numero di core della macchina su cui il WordleServer verrà eseguito. La FixedThreadpool è caratterizzata da una grandezza fissa del pool di thread, che porta nel nostro caso ad una corretta amministrazione di richieste quando il numero di client connessi non supera la grandezza del pool, evitando così context switch troppo frequenti. Diverso è il caso in cui il numero di client risulti troppo superiore al numero di thread, la gestione statica del pool potrebbe risultare una castrazione, ed il carico di ogni thread eccessivo per garantire un corretto parallelismo. Il WordleServer fornisce anche dei metodi al WordleClient che da remoto può invocare (JRMI), utili per le fasi di registrazione, login utente e per aderire al servizio di callback sul quale verranno ricevute notifiche al variare del podio.

FLUSSO DI ESECUZIONE E APPROFONDIMENTI

Una volta compilato il codice (vedi sezione “istruzioni per a compilazione”) ed eseguito in sequenza WordleServerMain e WordleClientMain i seguenti terminali si presenteranno così (un possibile flusso di esecuzione):

(I dati riportati dal gioco potrebbero non essere coerenti tra di loro essendo stati prelevati in più momenti e quindi in istanze di gioco diverse, risultano quindi non attendibili, ma abbastanza esplicativi sul fronte ‘istruzioni di gioco’ per mostrare uno scenario in cui un utente esegue tutti i possibili comandi in modo corretto. Infatti non sono stati presi in considerazione possibili inserimenti non corretti, nonostante il server ed il client siano stati programmati in modo da poter gestire possibili errori da parte dell’ utente utilizzatore ed imprevisti nell’ esecuzione)

Server:

GAME CREATED, NEW WORDLE'S WORD = depository ITALIAN TRASLATION: depositario  
READY

Il server dopo l’avvio legge dal file ServerConfigurationFIle.properties i parametri per le impostazioni iniziali. Una volta avviato WordleServer.start() crea un FixedThreadpool a cui verranno passati i task per il servizio dei client. Vengono anche creati Oggetti di tipo: Classification, RegistrationServiceImpl, NotificationServiceImpl, Game. Una volta creato il nuovo gioco restituisco la SecretWord e la sua traduzione in italiano. Crea dopo due skeleton per per i servizi di notifica e registrazione e li pubblica sul servizio di registry, inoltre controlla che l’ indirizzo di multicast sia disponibile.

Client:

-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-  
WELLCOME IN WORDLE  
IF YOU OWN AN ACCOUNT, WRITE 'login'  
ELSE, WRITE 'registration'

TO EXIT: 'exit'  
-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-

Il client dopo l’avvio legge dal file ClientConfigurationFile.properties i parametri per le impostazioni iniziali. Una volta avviato ClientServer.start() apre uno Scanner(System.in) e crea un Stub per la registrazione RMI. In questo momento client propone all’ utente di registrarsi oppure di effettuale la login.

>login  
DIGIT YOUR OWN USERNAME  
>raoul  
DIGIT YOUR OWN PASSWORD  
>12  
LOGIN SUCCESSFULLY  
REGISTERING FOR CALLBACK  
JOINED TO MULTICAST GROUP

CONNECTED TO THE SERVER

L’utente ha digitato ‘login’ quindi il client ha chiesto un username ed una password che vengono verificati tramite RMI nel DB delle registrazioni. L’utente viene riconosciuto e viene registrato al servizio di notifiche callback tramite RMI (variazioni podio), viene creato inoltre un thread gestore della creazione e del monitoraggio della MulticastSocket( share suggestions). Il client adesso si connette tramite una Socket al server e crea due stream wrapped DataOutputStream e DataInputStream.

Server:

USER LOGGED : raoul  
NEW CLIENT REGISTERED  
ACCEPTED CONNECTION FROM java.nio.channels.SocketChannel[connected local=/127.0.0.1:1039 remote=/127.0.0.1:54260]

Il server crea un thread per la serializzazione degli oggetti nel file RegistrationLog che conterrà tutti gli oggetti User, garantendo cosi la persistenza dei dati anche dopo la terminazione del server. Adesso il server è pronto per accettare richieste dai client, quindi, crea un ServerSocketChannel per l’ascolto, viene restituito un SocketChannel non bloccante e associato alla porta tcp. Crea, inoltre, un selettore per il multiplexing dei SocketChannel abilitati all’I\O e del SocketChannel che si occuperà delle Accept. Il MainThread entrerà in un ciclo while true per il monitoraggio dei canali. Se un client vuole connettersi, il selettore restituirà che il canale per le accept sarà pronto, crea una SocketChannel non bloccante e la registra in lettura sul Selector, associa come Attachment un oggetto Bytebuffer[] formato da due ByteBuffer che conterranno uno la grandezza di un intero ed uno la grandezza massima per un messaggio.

Client:

HELLO raoul  
-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-  
THESE ARE THE COMMANDS TO INTERACT WITH WORDLE  
TO PLAY: 'play'  
TO LOGOUT: 'logout'  
TO VIEW THE STATISTICS: 'statistics'  
TO VIEW THE CLASSIFICATION: 'classification'  
TO SHARE THE SUGGESTIONS: 'share'  
TO VIEW THE SUGGESTIONS YOU HAVE RECEIVED: 'sharing  
TO EXIT: 'exit'  
DO YOU NEED HELP? 'help'  
-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-  
>play  
MATCH STARTED

A questo punto il WordleClient ci propone una serie di istruzioni per poter interagire con esso ed entra in uno switchcase da cui esce solo e soltanto dopo aver digitato exit. L’utente scrive play per giocare una nuova partita quindi scriverà sul DataOutputStream un intero LenghMessage ed una stringa “username|password|play” e si metterà in lettura sull’BufferReader per la risposta del server.

Server:

raoul HAS STARTED A MATCH  
MESSAGE: raoul|play|ok  
SENT TO: /127.0.0.1:54260

Il Selector trova una SocketChannel pronta in lettura, quindi, restituisce un Set di chiavi pronte. Può adesso quindi andare a prendere dalla chiave il SocketChannel associato e l’oggetto Bytebuffer[] in attachment su di esso. Legge quindi dal SocketChannel e scrive sul BYtebuffer[], assicurandosi però di aver letto tutti i dati (essendo non bloccante la lettura) altrimenti finirà di leggere dopo il prossimo ciclo di monitoraggio. Se ha letto tutto trasforma in int ByteBuffer[0] ed in string ByteBuffer[1] adesso associa un nuovo ByteBuffer[] (libero) in attach alla SocketChannel. Per ottenere il formato voluto dal protocollo (“username|password|command”)tokenizza la String controllando tutti i campi che siano corretti. Uno switchcase valuterà il campo comando ed assegnerà il task al ThreadPool. Valutando il caso del comando ‘play’ nello specifico l’oggetto play controllerà se l’utente ha già giocato oppure viene creato una nuova istanza di Gioco per l’utente. Se tutto va a buon fine il thread invia sul SocketChannel un messaggio del tipo “username|command|exitcode”, anche se il SocketChannel non è bloccante il thread forza tramite un while la scrittura di tutta la stringa sul SocketChannel.

{

È stato descritto in modo dettagliato nei precedenti punti il tipo di protocollo usato e come client/server inviano/ricevono dati, da adesso verranno omessi i particolari ridondanti, quindi nelle seguenti illustrazioni useremo le parole ‘invia’,‘riceve’,‘legge’,’scrive’.

}

Client:

TO SUBMIT A WORD, DIGIT: 'submit' + 'word'  
>submit abjuratory

Da terminale viene proposto di sottomettere una parola usando il comando submit che viene inviata al server secondo il protocollo concordato.

Server:

MESSAGE: raoul|submit|tryagain|[?][F][F][F][?][?][?][?][T][T]  
SENT TO: /127.0.0.1:54260

Il server riceve il comando di submit ed assegna ad un thread il task submit. Recupera dal RegistrationDB (HashMap<username,User>) l’ User tramite l’ user recupera da Game il Match associato all’ User che contiene le informazioni sulla partita attuale quindi , lista dei suggerimenti, numero di suggerimenti e stato della partita in corso (win ,ended). Controlla se la parola che l’utente ha inviato è nel vocabolario e genera una stringa di suggerimenti (entrambi metodi offerti da Game), a questo punto esegue il test di correttezza sulla parola rilevando che (in questo caso) non corrisponde alla secret word. Adesso il server risponde al client con codice tryagain + la stringa di suggerimenti.

Client:

SUGGESTIONS: [?][F][F][F][?][?][?][?][T][T]  
BEEEEEEEP! WRONG WORD

TO SUBMIT A WORD, WRITE: submit + 'word'

Il client stampa i suggerimenti inviati dal server con l’esito della partita e invita l’ utente a sottomettere una nuova parola oppure a inviare un nuovo comando.

>share

L’ utente digita share per condividere i propri suggerimenti con il resto dei giocatori in partita, quindi invia al server un messaggio contenete il comando share per chiedere al server di condividere i propri suggerimenti che ha nel Match attuale.

Server:

MESSAGE: raoul|share|ok  
SENT TO: /127.0.0.1:54260  
SENT ON MULTICAST SOCKET:  
raoul  
1/12: [F][F][F][F][F][?][?][?][F][?]  
2/12: [F][F][F][F][F][?][?][?][F][?]  
3/12: [F][F][F][T][F][F][?][?][F][?]

Ricevuta la richiesta di share risponde al client come da protocollo, in seguito richiede il Match associato all’ User dalla classe Game in cui sarà contenuta la lista dei suggerimenti, che dopo aver creato un DatagramSocket ed aver impacchettato i dati già formattati dentro un DatagramPacket invia al gruppo di multicast.

Client:

>sharing  
raoul  
1/12: [F][F][F][F][F][?][?][?][F][?]  
2/12: [F][F][F][F][F][?][?][?][F][?]  
3/12: [F][F][F][T][F][F][?][?][F][?]

Come già detto in precedenza il client ha un thread dedicato al monitoraggio della MulticastSocket, il thread alla ricezione dei dati aggiunge alla lista dei messaggi Multicast ricevuti in precedenza anche i nuovi messaggi. Il client così quando l’utente digita sharing può aver accesso ad una lista di suggerimenti ricevuti che stampa a schermo togliendo ogni elemento visualizzato dalla lista.

>statistics  
match : 10  
win rate : 80.0  
current winning streak : 0  
max winning streak : 8  
distribution of attempt to win the game :   
+1 attempt : 50.0  
+2 attempt : 50.0  
+3 attempt : 0.0  
+4 attempt : 0.0  
+5 attempt : 0.0  
+6 attempt : 0.0  
+7 attempt : 0.0  
+8 attempt : 0.0  
+9 attempt : 0.0  
+10 attempt : 0.0  
+11 attempt : 0.0  
+12 attempt : 0.0

L’ utente digita ‘statistics’ per poter visualizzare le statistiche di tutti i Game a cui ha partecipato. Il client manda una richiesta al server con comando statistics per ricevere i dati, in questo caso non aspetta una risposta come da protocollo, ma aspetterà già la stringa di dimensione fissata avendo le statistiche associate all’ User tutte la stessa grandezza.

Server:

MESSAGE:   
match : 10  
win rate : 80.0  
current winning streak : 0  
max winning streak : 8  
distribution of attempt to win the game :   
+1 attempt : 50.0  
+2 attempt : 50.0  
+3 attempt : 0.0  
+4 attempt : 0.0  
+5 attempt : 0.0  
+6 attempt : 0.0  
+7 attempt : 0.0  
+8 attempt : 0.0  
+9 attempt : 0.0  
+10 attempt : 0.0  
+11 attempt : 0.0  
+12 attempt : 0.0  
SENT TO: /127.0.0.1:58105

Il server ricava l’User dal RegistrationDB da cui prenderà l’oggetto tramite un metodo get. È sato definito per le statistiche un metodo toString per convertire in stringhe i dati contenuti nell’ oggeto; quindi, vengono inviati i dati al client che sarà pronto a riceverli.

Client:

>classification  
1°:  
username: raoul  
score: 5.3333335  
statistics:  
match : 10  
win rate : 80.0  
current winning streak : 0  
max winning streak : 8  
distribution of attempt to win the game :  
+1 attempt : 50.0  
+2 attempt : 50.0  
+3 attempt : 0.0  
+4 attempt : 0.0  
+5 attempt : 0.0  
+6 attempt : 0.0  
+7 attempt : 0.0  
+8 attempt : 0.0  
+9 attempt : 0.0  
+10 attempt : 0.0  
+11 attempt : 0.0  
+12 attempt : 0.0

(in questa immagine è stato riportato solo il primo posto della classifica)

L’utente digita ‘classification’ per poter visualizzare la classifica di tutti i giocatori che si sono registrati al gioco. Il client come da protocollo invia la richiesta al server e attende i dati che verranno stampati come stringa.

Server:

MESSAGE: 1°:  
username: raoul  
score: 5.3333335  
statistics:  
match : 10  
win rate : 80.0  
current winning streak : 0  
max winning streak : 8  
distribution of attempt to win the game :  
+1 attempt : 50.0  
+2 attempt : 50.0  
+3 attempt : 0.0  
+4 attempt : 0.0  
+5 attempt : 0.0  
+6 attempt : 0.0  
+7 attempt : 0.0  
+8 attempt : 0.0  
+9 attempt : 0.0  
+10 attempt : 0.0  
+11 attempt : 0.0  
+12 attempt : 0.0

Il server alla ricezione de comando assegna il task specifico ad un thread che recupera Classifica dal RegistrationDB e tramite il metodo toString la converte in stringa e la invia al client che sta aspettando i dati.

Client:

>help  
-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-  
THESE ARE THE COMMANDS TO INTERACT WITH WORDLE  
TO PLAY: 'play'  
TO LOGOUT: 'logout'  
TO VIEW THE STATISTICS: 'statistics'  
TO VIEW THE CLASSIFICATION: 'classification'  
TO SHARE THE SUGGESTIONS: 'share'  
TO VIEW THE SUGGESTIONS YOU HAVE RECEIVED: 'sharing'  
TO EXIT: 'exit'  
DO YOU NEED HELP? 'help'  
-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-\_-

In ogni momento dopo aver eseguito il login, l’ utente che ha bisogno di un aiuto nell’ uso dei comandi può digitare ‘help’ per avere stampati a schermo tutto il set di comandi necessari per interagire con il WordleCLient.

>logout  
LOGOUT SUCCESSFUL

L’ utente può decidere di eseguire un logout senza perdere le statistiche del proprio User (essendo salvate dal WordleServer). Così facendo non potrà più interagire con il WordleCLient se non per chiudere il processo. Il client all’ esecuzione del programma invia come da protocollo una richiesta di ‘logout’ al server ed aspetta l’esito

Server:

USER: UNLOGGED raoul  
MESSAGE: raoul|logout|ok  
SENT TO: /127.0.0.1:62321

il server, ricevuto il comando di logout procede ad assegnare il task ad un thread che provvede a recuperare l’utente dal RegistrationDB e a settare l’attributo logged a false risponde al client come da protocollo. Prima di terminare la procedura termina il Match relativo all’ User (se non era già termianto) ed aggiorna le Statistics e la Classification.

Client:

>exit  
Process finished with exit code 0

L’utente digitando ‘exit’ avvia sul client una procedura di terminazione che chiude lo Stream(in), il Socket ed il MulticastSocket ed in seguito termina.

Client:

UPDATE EVENT: PODIUM HAS CHANGED  
[raoul, user3, user2]

Questo tipo di stampa si verifica quando il podio presente nella classe

ILLUSTRAZIONE DI METODI E STRUTTURE DELLE CLASSI

In questa sezione vedremo ogni entità in cui è suddiviso Wordle con descrizione dei principali metodi e delle strutture utilizzate.

WORDLE SERVER MAIN

È la classe che contiene il metodo main da cui il server viene avviato, si occupa reperire le informazioni registrate nel file “config/ServerConfigurationFile.properties” da cui recupera i seguenti campi: multicast\_address, multicast\_port, registry\_port, socket\_tcp\_port, tyme\_to\_awayt, registr\_obj, notific\_obj, time\_to\_refresh. Si occupa inoltre di creare un oggetto WordleServer, di passargli i dati recuperati dal file e di lanciare il metodo start() che avvierà il WordleServer.

WORDLE SERVER

È la classe cardine del nostro server occupandosi della maggior parte dei metodi e delle strutture dati. Nello specifico si occupa di orchestrare:

* FixedThreadPool: per la gestione in parallelo dei task che i client richiedono.
* RegistrationServiceImpl: per la registrazione e la gestinone degli utenti già registrati tramite RMI.
* Game: la classe che rappresenta l’istanza di gioco
* Classification: contiene in ordine la classifica utenti che hanno giocato a Wordle.
* NotificationServiceImpl: il servizio di notifica del podio tramite CallBack RMI ai client registrati.
* serverStatusSerializer: implementa tramite un thread la serializzazione dello stato degli utenti registrati dall’ avvio del server sul file “config/RegistrationLog.txt”.
* Select e ChannelSocket: per il multiplexing dei canali NIO non bloccanti su cui il client faranno le richieste e se cui il server risponderà.
* Multicast Group: su cui i server potrà inviare Datagrammi UDP per la condivisione dei suggerimenti ai client.

User

Rappresenta l’istanza utente in cui sono contenuti i principali dati utente come l’username(univoco), password, stato logged o unlogged e le Statistics. Mette a disposizione metodi get, set, toString ,equals, ed implementa Comparable.

Statistics

È legata ad ogni User e contiene l’insieme delle statistiche globali utili all’ utente: numero partite giocate, numero vittorie, win rate, win streak, max win streak, guess distribution. Quest ultima rappresenta la distribuzione delle vittorie sui tentativi, ed è implementato come float[12]. Ogni indice rappresenta la quantità di tentativi impiegati per una vittoria, nei valori è rappresentata la percentuale di volte in cui è stata vinta una partita con un determinato numero di tentativi. Mette a disposizioni metodi per l’ aggiornamento dei dati ogni volta che l’ utente sottomette una parola.