WORDLE

Un gioco di parole 3.0

Alessia Anile
matricola 619554

Indice

1 Introduzione e scelte personali

Ho implementato WORDLE tramite due applicazioni principali: una per avviare il server e l'altra per l'avvio di un client

Come prima cosa, è necessario avviare il server. Per farlo, si avvia la classe ServerMain che legge la configurazione dall'apposito file JSON, istanzia il socket TCP del server e lo pone in attesa di richieste da parte dei client. All'arrivo di nuove richieste, il server otterrà un nuovo socket e, sfruttando l'apposito Thread Pool, avvierà una nuova partita. La partita viene gestita dalla classe Sessione. Inoltre, la classe ServerMain avvia periodicamente un task per l'aggiornamento della Secret Word

A questo punto, per iniziare una nuova partita, bisogna avviare la classe ClientMain: dopo aver letto la propria configurazione come fa il server, chiede all'utente di inserire il comando numerico per scegliere l'azione.
L'utente può scegliere tra creare un nuovo account (registrazione), accedere a uno già esistente (login) oppure abbandonare.
Dopo la registrazione, deve comunque effettuare l'accesso all'account creato e, se questo va a buon fine, può inserire il prossimo comando, scegliendo tra:

- gioca
- visualizza le tue statistiche
- condividi le tue statistiche
- mostra i risultati dei miei amici
- logout

Per sessione s'intende da quando un utente esegue il login a quando esegue il logout, o la singola fase di registrazione. Durante una sessione autenticata, se la Secret Word cambia, l'utente può giocare per più di una Secret Word. Può richiedere la sospensione di una partita e riprenderla in seguito, ma se esegue il logout, al suo rientro non potrà più farlo, anche se la Secret Word non è ancora stata aggiornata. Prima di avviare la partita, dopo o durante la sospensione può richiedere di visualizzare le sue statistiche o di visualizzare i risultati dei suoi amici -gli utenti che appartengono al suo stesso gruppo sociale- ma non può condividere i propri risultati finchè non termina la partita (con esito positivo o negativo). I risultati degli amici che visualizza sono quelli condivisi da quando l'utente stesso ha eseguito il login

Dopo il logout, viene mostrato nuovamente il menù iniziale e l'utente può accedere a un altro account oppure registrarne uno nuovo, se così desidera

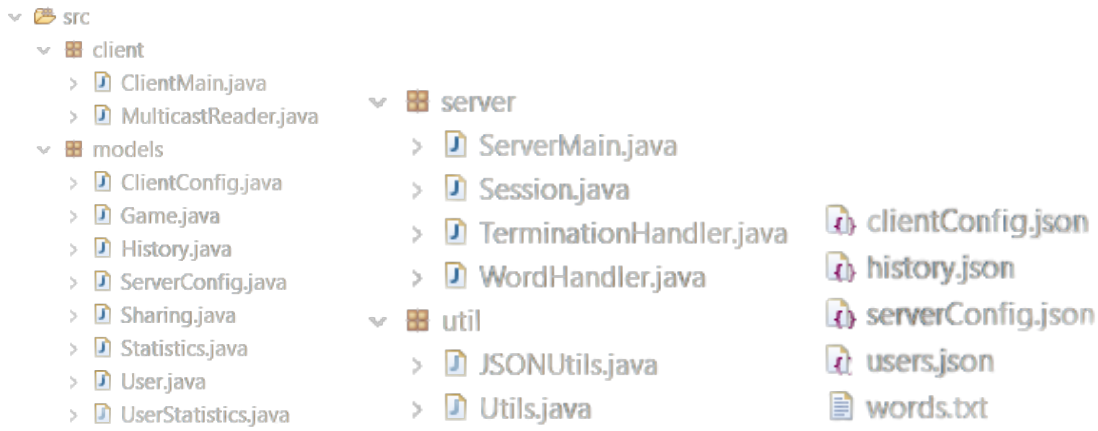
2 Overview della struttura

Il progetto è stato suddiviso in 4 package:

- server: contiene la classe Main per l'avvio dell'applicazione server, la sua gestione e gli handler necessari
- client: contiene la classe Main per l'avvio dei client e il task addetto alla ricezione da multicast.
- models: contiene i modelli utilizzati dal server e dai client per la gestione del gioco e per il salvataggio delle informazioni
- util: contiene classi con metodi di utilità comuni a server e client

Oltre ai package, contenuti nella cartella src, sono presenti:

- la cartella lib per eventuali librerie esterne
- i file di configurazione
- gli storici relativi alle Secret Words e agli utenti
- il file del vocabolario
- la cartella doc: aprendo il file index.html è possibile leggere la javadoc del progetto



3 Struttura del progetto

3.1 Librerie esterne

La cartella lib contiene il jar dell'unica libreria esterna utilizzata, Gson 2.10, per le operazioni su file JSON

3.2 File di configurazione

Sono presenti due file di configurazione in formato JSON: serverConfig.json per il server e clientConfig.json per il client

- serverConfig.json:
  - port: il numero di porta del server
  - multicastPort: il numero di porta usata per il multicast
  - multicastAddress: l'indirizzo del gruppo multicast a cui il server i messaggi UDP
  - updateFrequency: indica la frequenza di aggiornamento della Secret Word
  - updateFrequencyUnit: è l'unità di misura della frequenza di aggiornamento della Secret Word. Deve essere impostata a "m", "h" o "g", rispettivamente minuti, ore e giorni; se non impostata o contenente diverso valore, viene assunto come secondi
  - historyFile: nome del file JSON contenente lo storico delle Secret Words
  - vocabularyFile: nome del file contenente il vocabolario
  - usersFile: nome del file JSON contenente lo storico degli account registrati
- clientConfig.json:
  - serverAddress: indirizzo del server
  - port: numero di porta del server
  - multicastAddress: indirizzo del gruppo multicast da cui il client riceve i messaggi UDP
  - multicastPort: numero di porta multicast

3.3 Altri file

Sono presenti altri 3 file:

- words.txt: il vocabolario contenente tutte le parole ammesse da WORDLE
- history.json: il file JSON contenente lo storico di tutte le Secret Words. In particolare, contiene la Secret Word, la data e l'ora in cui è stata impostata e il numero progressivo corrispondente
- users.json: il file JSON contenente tutte le informazioni sugli account registrati dagli utenti. In particolare, sono presenti le seguenti informazioni:
  - username
  - hashedPassword: password dell'utente a cui è stata applicata una funzione hash per motivi di sicurezza
  - lastGameDate: data e ora dell'ultima partita
  - active: stato di attività dell'utente. False se l'utente è inattivo in quel momento, true altrimenti
  - stats: statistiche dell'utente. Comprende il numero di partite giocate, il numero di partite vinte (informazione non condivisa ma utile per calcolare i vari punteggi delle statistiche), la percentuale di partite vinte, la lunghezza dell'ultimo e del massimo streak di vittorie e un array contenente la guess distribution, in ogni partita vinta

3.4 Package

Elenco e descrizione delle classi presenti nei package e dei relativi metodi

3.4.1 Package models

3.4.1.1 ClientConfig Modello per la serializzazione e deserializzazione del file di configurazione del client

3.4.1.2 ServerConfig Modello per la serializzazione e deserializzazione del file di configurazione del server

3.4.1.3 Game Modello per la serializzazione e deserializzazione dell'oggetto "game" nel file di configurazione del server

3.4.1.4 History Modello per la serializzazione e deserializzazione del file contenente lo storico delle Secret Words

3.4.1.5 Statistics Modello per la serializzazione e deserializzazione dell'oggetto "stats" nel file JSON degli utenti

3.4.1.6 UserStatistics La classe UserStatistics rappresenta le informazioni pubbliche di un utente: il suo username e le sue statistiche

3.4.1.7 User La classe User estende UserStatistics e rappresenta tutte le informazioni di un utente, oltre a quelle rappresentate da UserStatistics. La separazione è stata effettuata per mantenere netta la distinzione tra informazioni pubbliche e private di un utente, quali:

- hashedPassword
- lastGameDate
- active

Se l'utente non ha mai giocato (e quindi lastGameDate non rappresenta effettivamente una data), questa viene inizializzata al 1-1-1970 per indicare tale informazione senza ricorrere a valori null che potrebbero causare problemi al momento del confronto di date

3.4.1.8 Sharing Dopo aver giocato, ogni utente può decidere di condividere le informazioni relative alla partita appena conclusa. Per farlo, la classe Sharing permette di ottenere un oggetto contenente

- wordNumber: numero della Secret Word a cui fa riferimento il risultato
- numTentativi: numero di tentativi effettuati
- indiziTot: ArrayList di stringhe in cui sono presenti gli indizi inviati di volta in volta da parte del server

3.4.2 Package util

3.4.2.1 JSONUtils La classe JSONUtils contiene metodi di utilità per operare con i file JSON

writeJsonFile
Esegue la serializzazione di un file JSON usando la libreria gson. È un metodo generico: per esempio, è stato usato sia per la serializzazione del file degli utenti che per il file della configurazione del server

readJsonFile
Esegue la deserializzazione di un file JSON usando la libreria gson

updateUsersFile
Esegue la scrittura della lista degli utenti sull'apposito file JSON. Prima di effettuare la scrittura effettua una lettura, sfruttata sia per non avere inconsistenze, sia per cercare se l'utente esiste già e quindi va semplicemente modificato oppure se deve essere aggiunto

3.4.2.2 Utils La classe Utils, come già specificato, contiene metodi di utilità comuni a client e server. Contiene anche le costanti per la definizione dei colori della console, usati in particolare per mostrare gli indizi all'utente durante il gioco e dopo la condivisione. Un'altra costante utile è DATE_FORMAT, che stabilisce un formato standard per la data (che evita anche problemi dovuti a configurazioni differenti dei sistemi operativi)

readFile
Esegue la lettura di un file riga per riga, restituendo un array di stringhe che le contiene tutte

read
Esegue una lettura dal socket usando un BufferedReader. Può sollevare eccezioni di due tipi:

- SocketException: nel caso di problemi relativi al socket, propago l'eccezione
- IOException: nel caso di problemi I/O, dopo 5 tentativi di lettura falliti, propago l'eccezione

write
Esegue una scrittura sul socket usando un BufferedWriter. Il tipo dell'oggetto da scrivere è generico, così da poter sfruttare questo metodo sia per scrivere interi che stringhe. Può sollevare eccezioni di due tipi:

- SocketException: nel caso di problemi relativi al socket, propago l'eccezione
- IOException: nel caso di problemi I/O, dopo 5 tentativi di scrittura falliti, propago l'eccezione

getNowDate
Restituisce un oggetto Date con data e ora attuali

getDateFromString
Effettua il parsing di una stringa per ottenere un oggetto di tipo Date, utilizzando il formato standard definito dalla classe. In caso di input non valido o vuoto, viene restituita la data 01-01-1970 01:00:00

getMillis
Restituisce il numero di millisecondi a partire da un oggetto di tipo Date

getUpdateFrequencyMillis
A partire dalla configurazione del server, con questo metodo si ottiene la frequenza di aggiornamento della Secret Word in millisecondi

Altri metodi non descritti ulteriormente sono newSocket, newReader, newWriter, closeSocket, closeReader e closeWriter

3.4.3 Package server

3.4.3.1 ServerMain La classe ServerMain avvia il server effettuandone la configurazione iniziale. Mantiene una ArrayList¡Socket¿ di client connessi, utile al termination handler. Per gestire i client che desiderano avviare una connessione, viene utilizzato un Cached Thread Pool, traendo vantaggio dalla sua elasticità.
ServerMain si occupa anche dell'esecuzione di WordHandler: sottomette periodicamente un task (scheduleAtFixedRate) per l'aggiornamento della Secret Word

disconnect
Rimuove il socket dalla struttura dati connectedClients

3.4.3.2 TerminationHandler La classe TerminationHandler gestisce la terminazione del server dopo una user interrupt, causata per esempio dalla combinazione di tasti ctrl+c, o all'invocazione di una System.exit()

run
Termina il server stesso, i client connessi e WordHandler

3.4.3.3 Session La classe Session gestisce la comunicazione e l'interazione con il client, lato server. In particolare, alcuni attributi rappresentano dei flag di stato:

- quit: true se l'utente ha deciso di sospendere il gioco, false altrimenti
- shared: true se l'utente ha già condiviso il suo risultato per l'ultima partita giocata, false altrimenti

In caso di errori, la sessione dell'utente termina comunque

run
Si inizia con la fase di autenticazione: il server riceve la stringa contenente username, password e comando, inseriti dall'utente. Per maggiore sicurezza, viene eseguito l'hashing della password con il metodo hashCode() [nonostante sia ritenuto poco sicuro perchè può restituire lo stesso hash per più stringhe] Il server esegue quindi

l'operazione richiesta e comunica l'esito. Una volta terminata la fase di autenticazione, il client comunica al server i comandi successivi e il server esegue le relative operazioni. Dopodichè il server aspetta un altro comando ma prima, se l'utente ha richiesto di giocare ed è stato autorizzato dal server (con il metodo playWordle), è necessario leggere di volta in volta le parole e inviare l'esito e/o gli indizi.
Se l'utente indovina la parola, vengono aggiornate le sue statistiche considerando la partita come vinta. Dopo 12 tentativi, la partita si considera persa: le statistiche vengono aggiornate di conseguenza.
Se il server riceve il comando 5 esegue il logout, interrompe la richiesta di nuovi comandi e la sessione viene terminata, anche in caso di errori durante il logout

register
Effettuare la registrazione lato server, terminando poi la connessione TCP instaurata. Aggiorna il file JSON degli utenti aggiungendo un nuovo utente o restituisce un valore di errore

login
Esegue l'accesso a un account già esistente, se le credenziali ricevute sono corrispondono a quelle registrate sul file JSON degli utenti e l'utente non risulta già attivo. Se l'operazione va a buon fine, prende il riferimento all'utente, con l'oggetto User player e il suo stato viene impostato come attivo

logout
Avviare la procedura di logout. Verifica che lo username passato come parametro sia effettivamente quello del player autenticato e restituisce un valore che indica se è l'operazione è andata a buon fine

playWORDLE
Gestisce la partita lato server, chiedendo di avviarla:

- Se la data e l'ora di aggiornamento della Secret Word precede la data e l'ora dell'ultima partita dell'utente e la partita non è stata sospesa (quit==false): fallimento
- Se quit==true, la partita è stata sospesa: resetto il flag e si distinguono altri due casi
  - La parola è cambiata: resetto la partita, iniziandone una nuova
  - La parola non è cambiata: la partita viene ripresa
- Negli altri casi, è possibile iniziare una nuova partita

La data e l'ora dell'ultima partita vengono aggiornate al momento attuale

receiveWords
Legge una parola inserita e richiede gli indizi

getIndizi
Restituisce una stringa contenente gli indizi relativi a una parola passata come parametro: un simbolo [+/?/X] per ogni lettera o un valore numerico per i casi quit ed errore

sendMeStatistics
Costruisce la stringa per inviare al client le statistiche aggiornate dell'utente: concatena username e gli altri elementi con il separatore ';'

statsToShare
Costruisce la stringa per condividere il risultato dell'ultima partita giocata. La stringa è formata da un'intestazione (numero della Secret Word e numero di tentativi) e un corpo (per ogni tentativo, i relativi indizi). Le due parti sono separate da ':', all'interno invece è stato usato il separatore ';'

share
Condivide su multicast i risultati della partita sul gruppo sociale, inviando un messaggio UDP. Se il risultato è già stato condiviso o l'utente non ha ancora terminato una partita durante questa sessione, restituisce un valore di errore.

terminateSessions
Termina la sessione di un utente e lo rende inattivo. Rimuove il socket dalla lista di socket client connessi al server. Aggiorna il file JSON degli utenti e infine chiude il socket.

updateStatistics
Ricalcola le statistiche dell'utente autenticato considerando se l'ultima partita è stata vinta (e quindi dopo quanti tentativi) o persa

3.4.3.4 WordHandler

La classe WordHandler gestisce il gestore dell'aggiornamento delle Secret Words

run
Metodo del task per effettuare periodicamente l'aggiornamento:
seleziona la nuova Secret Word e la imposta come tale, gestendo sia il file della configurazione del server che quello dello storico delle Secret Words

3.4.4 Package client

3.4.4.1 ClientMain

La classe ClientMain avvia un client: gestisce l'interazione con l'utente e la comunicazione con il server

main
Nel main viene letta la configurazione dall'apposito file e avviene poi la prima richie-

sta di comando all'utente. Se inserisce il comando 1 -corrispondente alla creazione di un nuovo account- viene invocato il metodo register, se viene inserito 2 viene invocato login e, invece, con 3 si termina l'esecuzione.
La fase di autenticazione termina quando si ottiene un socket dal metodo login, e a questo punto viene instaurata una connessione TCP persistente

insertCrendentials
Richiede le credenziali -username, password- all'utente.
Le credenziali, insieme al comando inserito inizialmente, vengono concatenate tra loro con il separatore ';'.
Username può contenere spazi, username può contenerli solo al suo interno; eventuali spazi in testa o in coda vengono rimossi

register
L'utente ha richiesto la creazione di un nuovo account.
Il metodo avvia quindi una connessione TCP (non persistente) con il server, gli invia le credenziali e legge l'esito. Chiude la connessione.
Esiti negativi possono essere causati dall'inserimento di password vuota o dalla scelta di uno username già in uso da altri utenti

login
L'utente ha richiesto l'accesso a un account già esistente. Il metodo avvia una connessione TCP persistente con il server, gli invia le credenziali e anche in questo caso legge l'esito inviato come risposta. Restituisce il socket per la connessione, null se l'esito è negativo.
Esiti negativi possono essere causati dall'inserimento di credenziali errate o dal tentativo di accedere un account che in quel momento è già in uso su un altro client

executeAuthenticated
Viene usato per l'esecuzione autenticata del client: usa il socket restituito da login per comunicare con il server. A questo punto il client si unisce al gruppo multicast e viene sottomesso il task per la ricezione multicast da tale gruppo.
Vengono proposti all'utente gli altri 5 comandi finchè non decide di eseguire il logout e, di conseguenza, abbandona il gruppo sociale

joinMulticast
Permette al client di unirsi al gruppo sociale. L'indirizzo del gruppo viene passato come parametro ed è stato letto in precedenza dal file della configurazione del client. Restituisce il MulticastSocket che partecipa al gruppo

initMulticastReader
Inizializza il MulticastReader avviando un nuovo Thread a cui sono passati come parametri il MulticastSocket necessario e il riferimento alla struttura dati sharings. Restituisce il riferimento al Thread

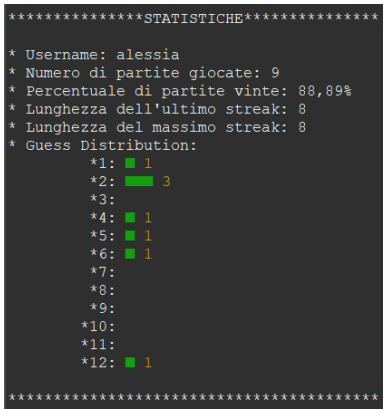


Figura 1: esempio output di printStats

Figura 2: esempio output di showMeSharing

leaveMulticast
Permette al client di abbandonare il gruppo multicast e termina di aspettare nuovi messaggi dal gruppo

printIndizi
Mostra all'utente gli indizi ricevuti dal server:

- Se l'indizio è '+': la lettera viene mostrata colorata di verde
- Se l'indizio è '?': la lettera viene mostrata colorata di giallo
- Se l'indizio è 'X': la lettera viene mostrata colorata di grigio

Se la parola era vincente, l'esito del server è formattato diversamente: è rappresentato da "0" che però viene sostituito da tanti + quante sono le lettere della parola (10), così da mostrarla interamente verde

printStats
Mostra all'utente le sue statistiche

showMeSharing
Mostra i risultati condivisi dagli utenti del gruppo multicast a cui l'utente appartiene. Oltre agli indizi, si mostrano il numero della Secret Word e il numero di tentativi effettuati

playWordle
Avvia il gioco lato client. Come prima cosa analizza l'esito inviato dal server e, se l'utente non ha già giocato per la Secret Word attuale, inizia una nuova partita. L'utente può decidere di sospendere la sua partita e riprenderla, mantenendola in sospeso fino a che non effettua il logout o fino a che la parola non cambia (in quest'ultimo caso, la partita viene resettata alla richiesta di giocare). Per richiedere la

sospensione è sufficiente inserire la parola "quit" al posto di una parola-tentativo. Dopo ogni tentativo, se la parola è contenuta nel vocabolario, non contiene spazi ed è della lunghezza corretta, vengono mostrati i relativi indizi; altrimenti un messaggio di errore

3.4.4.2 MulticastReader La classe MulticastReader rappresenta il thread del client che gestisce il socket multicast in ricezione. Mantiene la struttura dati sharings contenente i risultati ricevuti

run
Si occupa della ricezione multicast e aggiunge ciò che è appena stato letto, invocando il metodo save

save
In questo caso, i messaggi multicast rappresentano i risultati di partite terminate e condivise, perciò un questo metodo istanzia un oggetto di tipo Sharing a partire dalla stringa ricevuta. Infine, aggiunge l'oggetto alla struttura dati sharings

4 Codici di ritorno usati

register

- 2 → operazione andata a buon fine
- -1 → username già esistente
- -2 → password vuota

login

- 2 → operazione andata a buon fine
- -1 → credenziali errate
- -2 → sessione già esistente per questo utente

playWordle

- 1→ ok, inizia una nuova partita
- 2 → ok ma la parola è cambiata. Invece di riprendere la partita ne viene iniziata una nuova
- 3 → ok, la partita era stata sospesa e viene ripresa
- 0 → not ok, l'utente ha già giocato per quella Secret Word e la partita non può essere avviata
- -1 → errore

share

- 0 → l'utente non ha ancora terminato una partita in questa sessione e quindi non ha risultati da condividere
- 1 → operazione andata a buon fine
- 2 → il risultato dell'ultima partita della sessione dell'utente è già stato condiviso
- -1 → errore

logout, terminateSession

- 1 → operazione andata a buon fine
- -1 → errore. Nel caso di logout, username passato come parametro non corrisponde a quello dell'utente autenticato

getIndizi

- 0 → vittoria: la parola inserita corrisponde alla Secret Word
- 2 → la parola inserita non esiste nel vocabolario
- 3 → la lunghezza della parola inserita non è corretta
- 4 → la parola inserita contiene spazi
- 5 → quit: sospensione della partita
- Stringa contenente +/?/X: la parola inserita rispetta i requisiti ma non è la Secret Word
  - +: la lettera in questa posizione corrisponde alla lettera nella Secret Word
  - ?: la lettera in questa posizione è presente nella Secret Word ma in un'altra posizione
  - X: la lettera in questa posizione non è presente nella Secret Word

5 Thread attivati

Oltre ai main thread (un thread per il server e un thread per ogni client), vengono attivati i seguenti thread:

- Lato server
  - Session
  - WordHandler
  - TerminationHandler
- Lato client
  - MulticastReader: ogni client avvia un thread MulticastReader

6 Riepilogo delle strutture dati utilizzate

- Lato server
  - connectedClients (ServerMain, TerminationHandler): ArrayList contenente i socket dei client connessi al server
  - serverConfig (ServerMain): istanza della classe ServerConfig contenente le specifiche della configurazione del server
  - usersList (Session): ArrayList contenente le informazioni di tutti gli utenti registrati (User) Ogni thread istanzia la sua, quando necessario
  - indiziTot (Session): ArrayList contenente le stringhe di indizi per ogni tentativo, per l'ultima partita avviata dell'utente autenticato
  - historyList (WordHandler): ArrayList contenente le informazioni sulle Secret Words (History)
- Lato client
  - sharings (ClientMain, MulticastReader): ArrayList contenente i messaggi multicast ricevuti dal thread MulticastReader. Ogni sessione autenticata mantiene una propria ArrayList sharings. MulticastReader aggiunge elementi ma ClientMain non li rimuove, visualizza quelli aggiunti fino a quel momento
  - clientConfig (ClientMain): istanza della classe ClientConfig contenente le specifiche della configurazione del client

7 Sincronizzazione

Per l'accesso alle strutture dati condivise ho scelto di utilizzare un monitor. L'unica struttura dati condivisa è il file JSON degli utenti: per questo il metodo updateUsersFile deve essere synchronized, così che l'accesso avvenga da un thread alla volta, gli aggiornamenti siano consistenti e ogni utente aggiorni solo le proprie informazioni per quella sessione. Inoltre, prima di aggiornare il file, updateUsersFile lo legge nuovamente per essere sicuro di non perdere informazioni

Senza questa accortezza, se per esempio un utente esegue il login, inizia a giocare e nel frattempo un altro fa lo stesso e aggiorna il file, al momento del salvataggio da parte del primo, tutte le modifiche effettuate dal secondo vanno perse perchè non erano presenti in ciò che aveva letto inizialmente

Gli altri file JSON sono modificati e letti esclusivamente da un thread, così come le strutture dati private di ogni client (p.e. sharings), per esempio il file history.json può essere modificato solo da WordHandler

8 Manuale di istruzioni per gli utenti

8.1 Compilare

Per compilare il progetto su ambiente Windows:

Posizionarsi sulla root del progetto

```
mkdir binServer
```

```
mkdir binClient
```

```
javac -cp "lib\gson-2.10.jar;src" -d binServer src\server\ServerMain.java
```

```
javac -cp "lib\gson-2.10.jar;src" -d binClient src\client\ClientMain.java
```

Per compilare il progetto su ambiente Linux o macOS:

Posizionarsi sulla root del progetto

```
mkdir binServer && mkdir binClient
```

```
javac -cp "lib/gson-2.10.jar:src" -d binServer src/server/ServerMain.java
```

```
javac -cp "lib/gson-2.10.jar:src" -d binClient src/client/ClientMain.java
```

8.2 Eseguire

Per eseguire il progetto usando i file JAR forniti su ambiente Windows:

Posizionarsi sulla root del progetto

```
java -cp "lib\gson-2.10.jar;Server.jar" server.ServerMain
```

```
java -cp "lib\gson-2.10.jar;Client.jar" client.ClientMain
```

Per eseguire il progetto usando i file JAR forniti su ambiente Linux o macOS:

Posizionarsi sulla root del progetto

```
java -cp "lib/gson-2.10.jar:Server.jar" server.ServerMain
```

```
java -cp "lib/gson-2.10.jar:Client.jar" client.ClientMain
```