

# RELAZIONE WORDLE 3.0

LORENZO LUPETTI

## Definizione funzionamento, strutture dati scelte, componenti dell'applicazione e i thread attivi

La comunicazione client-server è implementata usando **JAVA I/O** e **threadpool**, utilizzando le classi `BufferedReader` e `PrintWriter` per leggere/scrivere messaggi.

Threadpool è implementato come `Executors.newFixedThreadPool(20)`, dato che vengono fatte operazioni di I/O e quindi il tempo di esecuzione di un thread è imprevedibile.

Se avessi molti client che si connettono e disconnettono subito potrebbe risultare nel consumo delle risorse dell'applicazione dato dall'overhead di creare molti thread.

### Server

Il server implementa la logica del gioco, la registrazione persistente degli utenti e la gestione di connessioni in arrivo da parte dei client.

Il server è composto dalle seguenti **classi**:

**WordleServerMain** è la classe principale contenente il metodo `main`, dove:

- legge dal file di configurazione `./server.properties` i dati del gruppo multicast, la porta dove accendere il servizio e la durata della parola segreta in secondi.
- legge i dati degli utenti salvati in `./usersData.json` e li salva in una `HashMap` con chiave `username` e valore `UserData`, classe che rappresenta i dati salvati di un utente come `username`, `password`, `vittorie`, `partite giocate`, `streaks` e `guess distribution` per calcolare lo score.
- legge il file delle parole dal file `./words.txt` e le salva in una `ArrayList`, non concorrente dato che verrà solo letta.
- crea il **SecretWordSessionManager**, che gestisce la parola segreta, descritto in seguito.
- registra un handler per gestire la terminazione del server definito dalla classe **ServerTerminationHandler**.
- crea la pool di thread e si mette ad accettare le connessioni dei client, gestite dal thread `ClientHandler`.

**ClientHandler** è il thread che gestisce la comunicazione con il client.

Implementa la logica del gioco lato server, rimanendo in attesa dei comandi da eseguire (come `login`, `playWordle...`) e inviando le risposte al client in base allo stato dell'applicazione. La risposta può essere un valore  $>0$ , che indica un errore, una keyword aspettata dal client oppure 0 se ok (in `sendWord()` ci sono più scambi di messaggi).

Gli vengono passati come argomenti:

- Socket su cui fare I/O che rappresenta la connessione con il client,
- La mappa degli utenti da cui legge e aggiorna i dati dell'utente in login(), register() e sendMeStatistics(),
- SecretWordSessionMenager da cui riceve info sulla parola segreta e la sessione dell'utente in playWordle() e sendWord(), il MulticastSocket dove inviare notifiche in share().

**SecretWordSessionMenager** è la classe da dove i ClientHandlers ricevono informazioni riguardo la sessione della parola segreta in corso, dove:

- Implementa la gestione della sessione della parola segreta, la creazione degli hints, sapere se una parola esiste o è la secret word, creare e rimuovere sessioni degli utenti.
- Ogni sessione è rappresentata dalla classe **SecretWordSession**, che contiene la parola segreta, l'intervallo di tempo in cui è attiva e la mappa delle sessioni degli utenti, salvati come ConcurrentHashMap, dato che i ClientHandlers creeranno, aggiorneranno ed elimineranno dalla mappa in modo concorrente, con chiave username e valore UserSession.

**UserSession** è la classe che contiene i dati relativi alla sessione di un singolo utente, quindi username, il numero di parole inviate, se ha vinto, condiviso e gli hints. Contiene funzioni per sapere se l'utente ha finito, costruire la notifica per sharing, aggiornare gli hints, sapere se ha vinto, finito, condiviso la partita e aggiornare il numero delle parole inviate.

- Alla creazione schedula un thread ogni tot secondi(durata parola segreta), definito come classe privata interna, **SecretWordChanger**, che quando attivato cambia la sessione corrente: cambia la parola segreta, resetta la mappa delle sessioni degli utenti e calcola il nuovo identificativo datainizio-datafine in base alla durata della parola segreta.

L'esecuzione ricorrente del thread è implementata tramite

Executors.newSingleThreadScheduleExecutor().scheduleAtFixedRate(SecretWordChanger, durataParolaSegreta).

**UserDataJsonWriter** è una classe, implementa come singleton, da cui l'applicazione può leggere e scrivere in maniera concorrente dal file json tramite i metodi sincronizzati readJsonMap() e writeJsonMap(Map) dell'istanza singleton. Per interagire con il file json utilizza la libreria esterna google.GSON.

Una volta eseguito il WordleServerMain, stamperà sul terminale informazioni come la parola cambiata, user che effettuano login/logout ed eventuali errori di connessione con client.

## Client

Il client si connette al servizio e comunica con il server, facendo giocare l'utente a Wordle tramite l'interfaccia grafica implementata tramite terminale.

La classe **WordleClientMain** contiene la maggior parte del codice, dove viene inizializzata la connessione, metodi per comunicare con il server e viene gestita l'interazione con l'utente tramite la visualizzazione dei menu e lettura dell'input:

- All'attivazione, vengono letti dal file di configurazione `./server.properties` i dati del gruppo multicast, la porta e ip del server.
- Avvia la connessione con il server tramite Socket e inizializza gli Stream I/O per inviare/ricevere messaggi.
- Entra nel loop principale, finché l'utente non richiede di uscire, composto da due fasi, login/register e menu:
  - login/register: fase iniziale in cui non sono loggato, quindi posso solo fare login/register/exit e viene mostrato il rispettivo menu.
  - menu: una volta effettuato il login viene mostrato il menu dove giocare, vedere statistiche, vedere notifiche, logout e uscire
  - I menù gestiscono e controllano l'input dell'utente e chiamano le rispettive funzioni richieste.
- Le singole funzionalità sono implementate tramite apposite funzioni che gestiscono la comunicazione con il server: `login()`, `register()`, `logout()`, `playWordle()`, `sendWord()`, `share()`, `showMeSharing()` e `sendMeStatistics()`.
- Le notifiche sono salvate in una `ArrayList`, non sincronizzata dato che il main legge e il thread collector scrive e basta.

L'unica altra classe è **MulticastNotifyCollector**, thread eseguito in parallelo, attivato al momento della login.

Gli viene passata per riferimento la lista delle notifiche, indirizzo e porta del gruppo multicast.

Una volta stabilita la connessione al multicast, si mette in ascolto di notifiche sul gruppo e ogni volta che ne arriva una nuova la aggiunge alla lista.

Si interrompe quando viene eseguito `logout()`: il server manda un pacchetto di terminazione per quel client in particolare, che gli altri scarteranno.

## Istruzioni di compilazione e avvio

### Server

Andare nella cartella `./WordleServer`.

Le configurazioni possono essere modificate nel file `./server.properties`, e sono:

- `serverport`: porta su cui attivare l'applicazione
- `servermulticast`: indirizzo ip del gruppo multicast

- portamulticast: porta del gruppo multicast
- wordduration: tempo tra una parola segreta e la prossima

**Compilazione:** `javac -cp ./gson-2.8.2.jar *.java`

**Esecuzione:** `java -jar WordleServer.jar`

Se è stato attivato con successo stamperà sul terminale: [SERVER] WordleServer started on port \_\_\_\_ e la parola segreta iniziale.

## Client

Andare nella cartella `./WordleClient`.

Le configurazioni possono essere modificate nel file `./client.properties`, e sono:

- server: hostname o ip del server
- portaserver: porta del server
- servermulticast: indirizzo ip del gruppo multicast
- portamulticast: porta del gruppo multicast

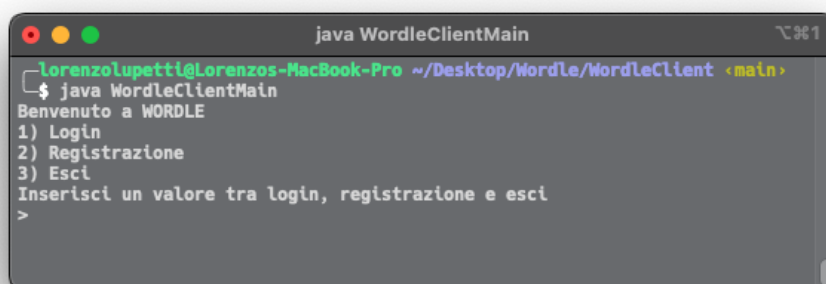
**Compilazione:** `javac *.java`

**Esecuzione:** `java -jar WordleClient.jar`

Da avviare dopo l' attivazione del server, se eseguito con successo e connesso al server stamperà sul terminale: Benvenuto a Wordle e il menu iniziale.

## Esempio di funzionamento del client

Appena accesa l'applicazione viene mostrato il primo menu:



```

lorenzolupetti@Lorenzos-MacBook-Pro ~/Desktop/Wordle/WordleClient <main>
$ java WordleClientMain
Benvenuto a WORDLE
1) Login
2) Registrazione
3) Esci
Inserisci un valore tra login, registrazione e esci
>

```

A questo punto, digitare login, registrazione oppure esci.

Dopo essersi registrati e/o loggati, viene mostrato secondo menu dove puoi giocare, vedere le statistiche, vedere notifiche, effettuare logout o uscire:

```
java WordleClientMain
Inserisci un valore tra login, registrazione e esci
>registrazione
Inserire username: lorenzo
Inserire password: lupetti
Registrazione avvenuta con successo!
1) Login
2) Registrazione
3) Esci
Inserisci un valore tra login, registrazione e esci
>login
Inserire username: lorenzo
Inserire password: lupetti
Login avvenuta con successo!
1) Gioca
2) Statistiche
3) Sharing
4) Logout
5) Esci
Inserisci un valore tra gioca, statistiche, sharing, logout, esci
>
```

Esempio di vittoria e successiva condivisione della partita:

```
java WordleClientMain
Login avvenuta con successo!
1) Gioca
2) Statistiche
3) Sharing
4) Logout
5) Esci
Inserisci un valore tra gioca, statistiche, sharing, logout, esci
>gioca
Puoi iniziare a mandare parole...(EXIT per uscire)
Inserire la parola da provare (lunga 10) (exit per uscire): philppist
philppist
Inserire la parola da provare (lunga 10) (exit per uscire): epizoicide
philppist
epizoicide
Inserire la parola da provare (lunga 10) (exit per uscire): analysable
philppist
epizoicide
analysable
Inserire la parola da provare (lunga 10) (exit per uscire): parastatic
philppist
epizoicide
analysable
parastatic
Hai indovinato la parola segreta!!!
Vuoi condividere la partita appena vinta?
(inserire sì oppure 1 per condividerla)>sì
Hai condiviso la partita con successo!
Lorenzo, ecco le tue statistiche aggiornate all'ultima partita:
PartiteGiocate: 1
Percentuale vittoria: 100.0%
Streak vittorie in corso: 1
Streak vittore massima: 1
Punteggio WAS: 5.000
```

Link GitHub: <https://github.com/Lupetti-Lorenzo/Wordle>