Università di Pisa - Dipartimento di Informatica

WORDLE : un gioco di parole 3.0

Progetto di Laboratorio III di Reti in Java

Autore : Stefanelli Alessia
Matricola : 615430
Anno Accademico 2022 - 2023

Table des matières

1 Introduzione

Nel progetto è stata implementata una versione semplificata del gioco Wordle. In questo gioco, i giocatori devono indovinare una parola segreta di 10 lettere in 12 tentativi al massimo e per poter rigiocare a Wordle devono aspettare la parola segreta successiva. Per ogni giocatore vengono registrate le statistiche di gioco.
Per implementare il gioco, è stato usato un modello client-server con connessione TCP :
- il client gestisce l'interazione con l'utente.
- il server coordina il gioco e i dati degli utenti. Viene usata una struttura master-worker per gestire più client.
La parte social del gioco, ovvero dove i giocatori condividono i risultati delle partite, è realizzata mediante l'uso di un gruppo multicast a cui partecipano tutti i giocatori connessi al gioco.

2 Struttura generale del progetto

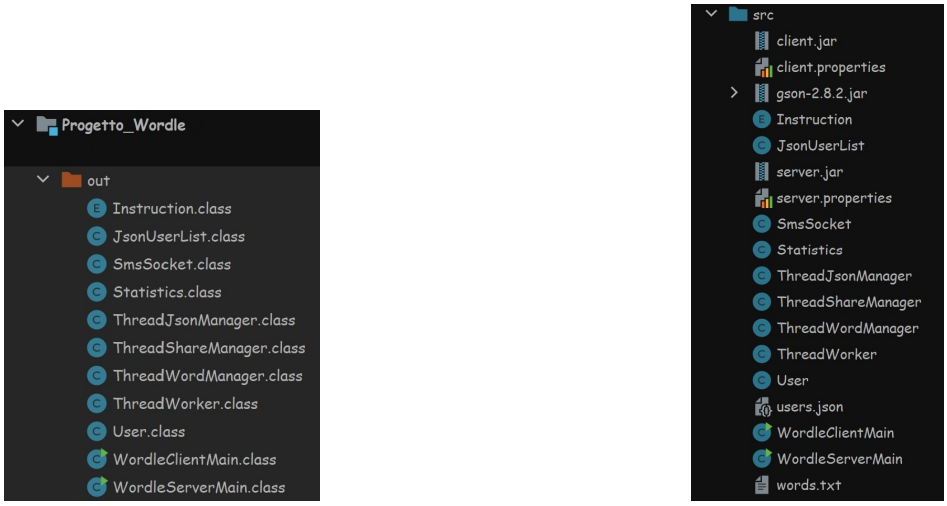


FIGURE 1 – struttura cartelle progetto

Nella cartella del progetto "Progetto_Wordle" ci sono 2 cartelle :
- In **src** sono presenti tutti i file relativi al gioco Wordle inclusi libreria e eseguibili jar (*.java, *.properties, words.txt, users.json, *.jar).
- In **out** sono presenti i file di classe (.class), creati durante la compilazione.

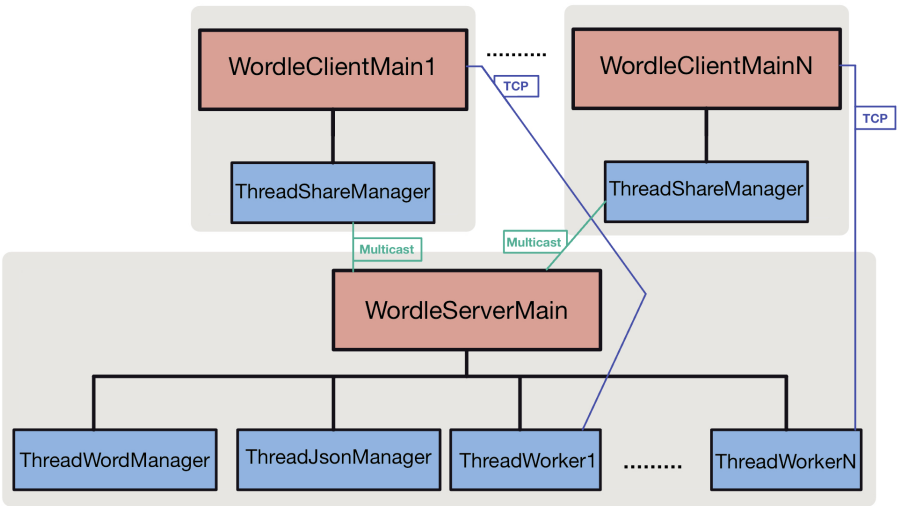


FIGURE 2 – Schema generale dei thread attivi

Nella *figura 2* sono rappresentati i thread attivati sia lato server che lato client. La connessione TCP client-server è rappresentata in blu, mentre la connessione UDP multicast in verde.

2.1 Lato Client

Nel lato client, il gioco è strutturato in diverse sezioni corrispondenti alle diverse fasi di interazione dell'utente :

- **Login/register room** : Questa è la sezione iniziale del gioco in cui il giocatore può :
  1. Connettersi con un utente già esistente (`login <username> <password>`)
  2. Registrarsi come un nuovo utente (`register <username> <password>`)
  3. Richiedere il manuale d'istruzione generale del gioco Wordle (`help`)
- **Service room** : In questa sezione, l'utente può scegliere tra i vari servizi offerti. Tra questi, i comandi `showMeSharing` e `help` sono gestiti direttamente dal client, mentre gli altri sono gestiti dal server. Ecco quali sono :
  1. Iniziare una partita (`play`)
  2. Richiedere le statistiche di gioco aggiornate (`SendMeStatistics`)
  3. Richiedere le partite condivise dagli altri giocatori(`showMeSharing`)
  4. Disconnettersi (`logout`)
  5. Richiedere il manuale d'istruzione generale del gioco Wordle (`help`)
- **Playing room** : In questa sezione di gioco, l'utente può richiedere di :
  1. Inviare una parola (`send <word>`)
  2. Condividere i risultati della partita (azione possibile alla fine della partita e non prima) (`share`)
  3. Disconnettersi (in questo caso la partita è persa) (`logout`)
  4. Richiedere il manuale d'istruzione generale del gioco Wordle (`help`)

Per gestire il lato client vengono usati i seguenti elementi [1] :

1. *client.properties*
2. *WordleClientMain*
3. *ThreadShareManager*

2.1.1 client.properties ↑

Il file "client.properties" è un file di configurazione utilizzato per impostare vari parametri all'avvio del programma client. Il file contiene le seguenti informazioni :

- `port=4000` : Specifica la porta sulla quale il client cercherà di connettersi al server.
- `ip_adress=127.0.0.1` : Indica l'indirizzo IP del server a cui il client si connetterà.
- `multicastIP=224.0.0.1` : Fornisce l'indirizzo IP multicast che il client userà per ricevere messaggi multicast.
- `multicastPort=4001` : Indica la porta sulla quale il client ascolterà i messaggi multicast.

1. La lista delle classi lato client hanno un link alla loro sotto-sezione corrispondente. La freccia ↑ nelle sotto-sezioni serve per ritornare alla lista delle classi lato client.

2.1.2 WordleClientMain.java ↑

Nel gioco Wordle, questa classe svolge il ruolo centrale per tutto ciò che riguarda il client. Alla sua esecuzione legge il **file di configurazione** "client.properties" (con `readConfig()`) per sapere a quale indirizzo IP e porta dovrà connettersi sia per la connessione TCP che multicast. Una volta che il client è configurato, fa partire il gioco ed entra nella sezione "**login/register room**" (vedi *figura 3*), che offre all'utente la possibilità di chiedere al server di registrarsi o connettersi al gioco. In entrambi i casi stabilisce una connessione TCP con il server. Invece per stabilire una connessione multicast è necessario effettuare il login.
Il client e server, per comunicare, usano un oggetto di tipo `SmsSocket`, che sarà approfondito nella sezione *2.3 Comunicazione*.

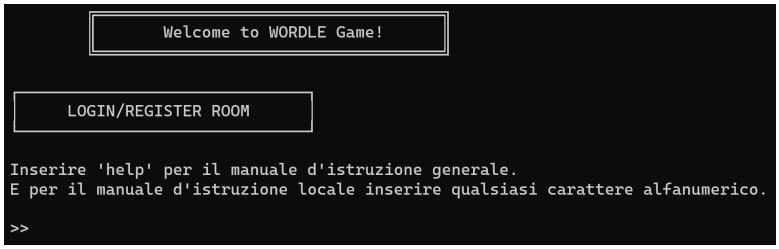


FIGURE 3 – Sezione "login/register room"

Dopo il comando `login` e a seguito della risposta dal server con esito positivo (`codice=7`), l'utente può entrare nella sezione "**service room**" (vedi *figura 4*), dove il client resta in attesa dell'input dell'utente relativa a uno dei comandi/servizi proposti (come *precedentemente* illustrato).

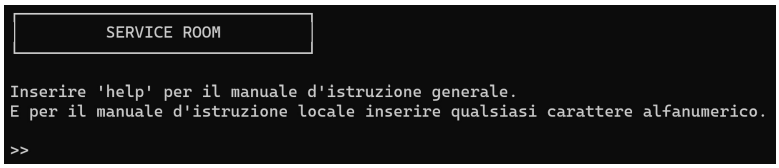


FIGURE 4 – Sezione "service room"

Se l'utente decide di eseguire il comando `play` per iniziare una partita e il server verifica che l'utente non abbia ancora affrontato la parola segreta del giorno, il client porterà l'utente nella sezione "**playing room**" (vedi *figura 5*), dove potrà selezionare uno dei comandi proposti (come *precedentemente* illustrato). Per quanto riguarda la rappresentazione dello status delle lettere della parola proposta è stato scelto di utilizzare i simboli `X, ?, +`, al posto dei colori.

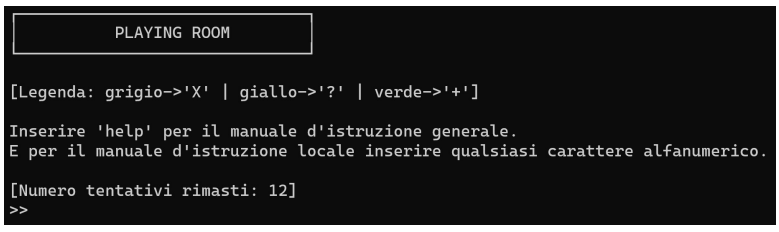


FIGURE 5 – Sezione "playing room"

L'utente può vedere le sue **statistiche** (vedi *figura 6*) aggiornate sia dopo la fine di una partita sia su richiesta diretta al server con il comando `sendMeStatistics` (nella sezione "service room").

```
------- Ecco le statistiche personali aggiornate -------
 Numero partite giocate : 3
 Numero partite vinte : 2
 Percentuale partite vinte : 66 %
 Ultima sequenza continua di vincite : 0
 Massima sequenza continua di vincite : 2
 Guess Distribution : [8, 1]
-------------------------------------------------------
```

FIGURE 6 – Statistiche dell'utente

Alla fine di una partita, il client chiede all'utente se vuole **condividere** il risultato della partita appena giocata con tutti i giocatori connessi (vedi *figura 7*). Nel caso di risposta positiva, il client invia al server la richiesta di condivisione del risultato, attivando il metodo `share()`.

```
Vuoi condividere la tua partita: si [s] o no [n] -> n
```

FIGURE 7 – Richiesta condivisione

Nella sezione "service room", l'utente ha la possibilità di richiedere i **risultati delle partite condivise** dagli altri giocatori, finora raccolti in una struttura dati, utilizzando il comando `showMeSharing` (vedi *figura 8*). Questa parte viene gestita da un altro thread (ThreadShareManager), che verrà approfondito dopo (nella sotto-sezione *2.1.3 ThreadShareManager.java*).

```
>> showMeSharing
[Legenda: grigio->'X' | giallo->'?' | verde->'+']


----- Username : caio -----
     Wordle 13 : 8/12
    ? X X X X ? X + ? X
    X X X ? X ? ? + + +
    X X ? X ? ? X + + +
    X X X ? X X + + + +
    X X X X X ? + + + +
    ? X X + X ? + + + +
    X X X + + X + + + +
    + + + + + + + + + +
```

```
----- Username : tizio -----
     Wordle 56 : 12/12
    X X X ? X ? X ? ? ?
    X ? X ? X ? X ? X ?
    X ? ? X ? ? ? X ? X
    X ? ? X ? X X X ? X
    X X X ? ? X + + X X
    ? X X X X + + + X ?
    X X ? X ? + + + ? X
    X X ? ? X + + + X X
    X X + + X + + + X ?
    X X X + + + + + + +
    X ? X + + + + + + +
    X X ? ? ? ? ? ? ? ?


----- Username : caio -----
     Wordle 111 : 1/12
    + + + + + + + + + +
```

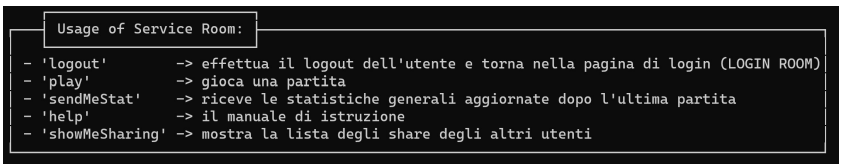
FIGURE 8 – Alcuni esempi di partite condivise

Nel caso l'utente usi il comando `logout` per **disconnettersi**, il client si occupa di chiudere la connessione e settare le variabili per ritornare alla sezione "login/register room". L'utente può eseguire il logout in 2 momenti diversi :

- durante una partita. Se l'utente decide di uscire, la partita verrà automaticamente registrata come persa. Questo meccanismo previene che l'utente esca e rientri nella speranza di tentare nuovamente con la stessa parola segreta ;
- quando si trova nella sezione "service room"

Per quanto riguarda il manuale di gioco, sono state implementate due versioni : una semplificata e una completa. Per quanto riguarda la ***versione semplificata***, ogni sezione dispone di un proprio **manuale d'istruzione**, specifico solo ai comandi eseguibili in quella sezione (vedi *figura 9*).

```
 Usage of Login Room:
| - 'help'                          -> il manuale di istruzione generale
| - 'register <username> <password>' -> registrazione dell'utente
| - 'login <username> <password>'    -> utente accede al servizio (SERVICE ROOM)
```



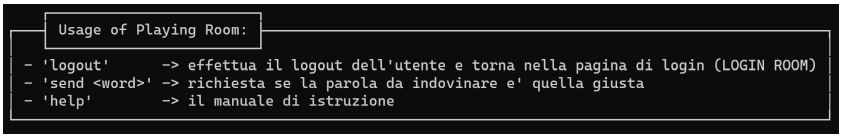


FIGURE 9 – Manuale d'istruzione per ogni sezione

Per quanto riguarda la ***versione completa***, il **manuale di istruzione generale** permette di avere una visione d'insieme di come è strutturato il gioco wordle (vedi *figura 10*).

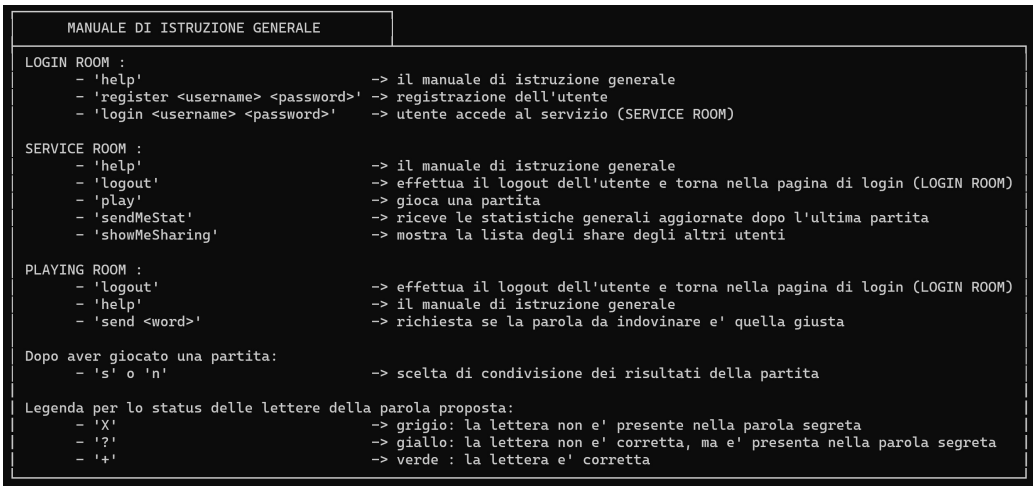


FIGURE 10 – Manuale d'istruzione generale

2.1.3 ThreadShareManager.java ↑

La classe `ThreadShareManager` opera come un thread indipendente e gestisce l'aspetto social del gioco. Essa consente agli utenti di ricevere, in tempo reale tramite una connessione multicast, le notifiche relative ai risultati delle partite condivise dagli altri giocatori.

L'utente entra a far parte di un gruppo multicast solamente dopo aver effettuato la connessione con il comando "`login <username> <password>`". Il ciclo di vita del thread si conclude quando l'utente sceglie di disconnettersi con il comando `logout`, indipendentemente dal fatto che si trovi in partita o nella sezione "service room".

E' stata utilizzata la struttura dati arraylist "`ArrayList<String> sharedGameList`" per tenere traccia dei risultati delle partite condivise dai giocatori.

Il metodo `logout()` serve per terminare il thread `ThreadShareManager`.

Il metodo `printSharedGameList()`, attivato dal comando `showMeSharing`, visualizza tutte le partite condivise nell'arraylist, evitando di mostrare le partite condivise dall'utente stesso e, successivamente, svuota la lista dai risultati visualizzati.

2.2 Lato Server

Per gestire il lato server vengono usati i seguenti elementi [2] :

1. *server.properties*
2. *Statistics*
3. *User*
4. *JsonUserList*
5. *ThreadJsonManager*
6. *ThreadWordManager*
7. *WordleServerMain*
8. *ThreadWorker*

2.2.1 server.properties ↑

Il file "server.properties" è un file di configurazione utilizzato per impostare vari parametri all'avvio del programma server. Il file contiene le seguenti informazioni :

- `port=4000` : specifica la porta sulla quale il server ascolterà le connessioni in arrivo dai client.
- `jsonFile=users.json` : indica il file JSON che verrà utilizzato per memorizzare le informazioni sugli utenti. Quindi è il Database del server.
- `fileTxt=words.txt` : è il file di testo che contiene le parole utilizzate nel gioco.
- `delayWord=5` : indica il ritardo (in secondi) con cui la parola del gioco cambierà.
- `delayJson=2` : indica l'intervallo di tempo (in secondi) con cui il file JSON (database) degli utenti viene aggiornato.
- `multicastIP=224.0.0.1` : fornisce l'indirizzo IP multicast che il server userà per inviare messaggi multicast.
- `multicastPort=4001` : indica la porta sulla quale il server invierà messaggi multicast.

2.2.2 Statistics.java ↑

La classe `Statistics` è stata progettata per tenere traccia delle prestazioni di un giocatore. Questa classe fornisce una struttura dati che registra varie metriche relative al rendimento del giocatore, come il numero di partite giocate, il numero di partite vinte, la serie attuale di vittorie consecutive (`currentStreak`) e la serie massima di vittorie consecutive raggiunta (`maxStreak`). Inoltre, mantiene una lista chiamata `guessDistribution` che tiene traccia del numero di tentativi impiegati per vincere ciascuna partita.
La classe offre una serie di metodi per accedere e aggiornare queste statistiche.

2.2.3 User.java ↑

La classe `User` permette di mantenere e gestire le informazioni associate a ciascun utente.

È stato scelto di conservare un elenco di tutte le parole segrete affrontate dall'utente (sia quelle vinte che perse) dal momento della sua registrazione, utilizzando la struttura `Vector<String> wordPlayedList`.

2. La lista delle classi lato server hanno un link alla loro sotto-sezione corrispondente. La freccia ↑ nelle sotto-sezioni serve per ritornare alla lista delle classi lato server.

2.2.4 JsonUserList.java ↑

La classe `JsonUserList` è un **monitor** per la gestione del database degli utenti. Questa classe utilizza la libreria Gson per serializzare e deserializzare gli oggetti `User` in formato JSON, permettendo così di salvare e recuperare le informazioni degli utenti nel file users.json.

Il costruttore della classe inizializza :

- una variabile `jsonFile` che rappresenta il percorso del file users.json ;
- un lista condivisa `Vector<User> userList`, che è un vettore destinato a contenere tutti gli utenti registrati (thread safe).

La classe offre una serie di metodi sincronizzati per manipolare e accedere alla lista degli utenti in maniera concorrente e sicura.
Tra i metodi offerti, due sono fondamentali per assicurare la persistenza dei dati :

- `saveUserListInJsonFile()` : questo metodo sincronizzato salva l'elenco corrente degli utenti nel file users.json, garantendo che un solo thread possa scrivere nel file in un dato momento ;
- `createUserListFromJsonFile()` : al lancio del server (attraverso il thread WordleServerMain), questo metodo legge il file users.json e riempie la lista degli utenti. Se il file risulta essere vuoto, viene inizializzato una nuova lista vuota.

2.2.5 ThreadJsonManager.java ↑

La classe `ThreadJsonManager` è responsabile dell'aggiornamento periodico del file users.json, che contiene la lista degli utenti registrati al gioco. Questo è un thread separato che si esegue all'avvio del programma server.

Il costruttore della classe accetta come parametro `jsonUserList`, che è un oggetto condiviso che rappresenta la lista degli utenti (istanze della classe `User`). Questa lista viene poi utilizzata per salvare gli utenti nel file users.json. Il salvataggio viene effettuato periodicamente attraverso il metodo `saveUserListInJsonFile()` della classe `JsonUserList`.

`jsonUserList` viene inizializzato nel thread `WordleServerMain` e utilizzato da tutti i thread worker.

Nel file di configurazione del server, la variabile `delayJsonUpdate` dovrebbe avere un valore ragionevolmente basso per garantire che il database venga aggiornato costantemente ad ogni piccola variazione. Questo assicura la persistenza delle informazioni relative agli utenti tra un avvio del server e il successivo.

2.2.6 ThreadWordManager.java ↑

La classe `ThreadWordManager` è responsabile dell'aggiornamento periodico della parola segreta utilizzata nel gioco Wordle. Questa classe è progettata per essere eseguita come un thread separato e ha il compito di estrarre una nuova parola dal file words.txt a intervalli regolari, che sono definiti dal parametro `delayWordUpdate` impostato nel file di configurazione "server.properties".

La classe gestisce due variabili importanti per il controllo della parola segreta e del suo aggiornamento costante :

- `wordListExtracted` : è un vettore che tiene traccia delle parole estratte dal file words.txt durante la sessione corrente del server ;

– `newWord` : è stata implementata come una variabile condivisa di tipo `AtomicReference<String>` per garantire l'aggiornamento thread-safe della stessa. Questo assicura che, anche nel caso in cui più thread tentino contemporaneamente di leggere (come nel caso dei `threadWorker`) o scrivere (come nel caso del `threadWordManager`) sulla stessa variabile, non si verifichino problemi di concorrenza.

Vengono inizializzate entrambe nella classe `WordleServerMain`, cosi che, oltre al `ThreadWordManager`, anche i threadWorker possano accedere al contenuto di esse.

Quando la classe `ThreadWordManager` viene eseguita, apre il file words.txt e calcola il numero totale di parole presenti nel file (`int countWords = (int) (fileReader.length() / 11)`). Successivamente, genera un numero casuale e utilizza questo numero per posizionarsi in un punto specifico del file (`seek`) e leggere una parola. Questa parola viene successivamente configurata come `newWord` e inserita all'interno della lista `wordListExtracted`.

`RandomAccessFile` è stato utilizzato per poter accedere direttamente a una posizione specifica nel file.

2.2.7 WordleServerMain.java ↑

Nel gioco Wordle, questa classe svolge il ruolo centrale per tutto ciò che riguarda il lato server. In particolare, è responsabile per l'inizializzazione e la gestione di vari componenti chiave del server, tra cui la connessione ai client, la gestione delle parole del gioco e la gestione degli utenti.

Alla sua esecuzione legge il **file di configurazione** "server.properties" (con `readConfig()`) per sapere :

– a quale porta della connessione TCP dovrà rimanere in ascolto
– su quale porta e indirizzo IP della connessione multicast dovrà inviare i risultati delle partite condivise
– i tempi di aggiornamento per le parole e gli utenti
– il file da dove leggere le parole e il file che funge da database di tutti gli utenti.

In seguito, il server popola una lista (`ArrayList<String> totalWordList`) con tutte le parole contenute nel file **words.txt**. Questa lista viene utilizzata durante la partita per verificare l'esistenza delle parole proposte dall'utente. La scelta di implementare questa lista utilizzando `ArrayList`, anziché `Vector`, è dovuta al fatto che la lista non subirà modifiche e sarà solamente letta da più thread.

Nel main thread del server vengono **inizializzati** i seguenti **elementi condivisi** :

– `jsonUserList` → è un'istanza della classe `JsonUserList` che gestisce la lista degli utenti ;
– `wordListExtracted` → è un vettore che tiene traccia delle parole estratte ;
– `connectedUser` → è un vettore che monitora gli utenti connessi al server ;
– `multicastSocket` → è un istanza di `DatagramSocket` usata per inviare i risultati delle partite al gruppo multicast degli utenti attivi ;
– `newWord` → è una variabile condivisa thread-safe che contiene la parola segreta.

Questa configurazione permette a tutti i thread worker di accedere allo stesso contenuto. Gli oggetti `jsonUserList`, `wordListExtracted` e `newWord` sono inoltre utilizzati dai rispettivi thread indipendenti `ThreadJsonManager` e `ThreadWordManager`.

Una volta impostato e avviato il server, si mette in attesa delle **connessioni dei client** su una specifica porta. Una volta stabilita la connessione, avvia un nuovo `ThreadWorker` per gestire l'interazione con il client.

La classe `WordleServerMain` utilizza uno **Scheduled Executor** (`Executors.newScheduledThreadPool(1)`) per eseguire i thread `ThreadJsonManager` e `ThreadWordManager` a intervalli regolari.

Inoltre utilizza un **Cached Executor** (`Executors.newCachedThreadPool()`) per gestire il pool di thread dei client connessi. Questa scelta è motivata dal fatto che, essendo il numero di client variabile, `newCachedThreadPool()` consente di gestire dinamicamente la creazione di nuovi thread quando è necessario. I thread inattivi vengono eliminati dopo un certo periodo di inattività, ottimizzando così l'utilizzo delle risorse. Questa soluzione è preferibile ad un pool di thread che alloca un numero fisso di thread worker, poiché questo può risultare insufficiente in caso di picchi di richieste o eccessivo se il carico di lavoro è basso.

In caso di **eccezioni** come `SocketException`, la classe `WordleServerMain` salva la lista degli utenti nel file users.json prima di terminare l'esecuzione.

Per la gestione della **connessione multicast**, è stato adottato un approccio centralizzato nel thread principale del server, utilizzando un socket UDP (`DatagramSocket`). Questa scelta permette al server di gestire un'unica connessione per trasmettere i risultati delle partite dal worker al gruppo multicast, al quale i client sono connessi.

2.2.8 ThreadWorker.java ↑

La classe ThreadWorker è responsabile della gestione dell'interazione tra il server e un singolo client, processando le richieste ricevute.

Ogni volta che un client si connette al server, viene creata una nuova istanza di `ThreadWorker` per gestire la comunicazione con quel client specifico. Il metodo `run()` del thread gestisce il ciclo di vita della connessione, ascoltando le richieste in arrivo e indirizzandole ai metodi appropriati per l'elaborazione.

Nel metodo `share()`, che consente all'utente di condividere i risultati della sua partita recente, è stata implementata la sincronizzazione mediante `synchronized(multicastSocket)`. Questo assicura una condivisione sicura dei risultati tra più threadWorker verso lo stesso gruppo multicast. Questo passaggio è necessario in quanto la connessione multicast (unica per tutti i workers) non è thread-safe.

Nel metodo `login()`, che gestisce la connessione di un utente su richiesta, è stato implementato `synchronized(connectedUser)` per assicurare una manipolazione sicura di `connectedUser`. Questo è essenziale dato che vengono eseguite operazioni composte, come `contains()` seguito da `add()`.

Nel metodo `register()`, che gestisce la registrazione di un utente su richiesta, è stato implementato il meccanismo `synchronized(jsonUserList)`, dato che vengono eseguite sequenzialmente operazioni composte sul vettore `jsonUserList`, quali : `isEmptyJsonUserList(), checkUsername(username)` e `insertUser(user)`.

2.3 Comunicazione

Per la comunicazione TCP tra client e server sono state implementate le seguenti classi :

1. *Instruction*
2. *SmsSocket*

2.3.1 Instruction.java ↑

La classe `Instruction` è definita come un tipo enumerativo (enum), che include un insieme di costanti che rappresentano l'azione specifica che il client desidera eseguire sul server.
E' stato necessario implementare `Serializable`, poiché facendo parte dell'oggetto `SmsSocket`, deve poter essere inviato attraverso la connessione di rete TCP.

2.3.2 SmsSocket.java ↑

La classe `SmsSocket` è progettata per definire il formato dei messaggi da trasmettere e implementa l'interfaccia `Serializable` per garantire la trasmissibilità degli oggetti di questa classe via TCP. Questa configurazione semplifica lo scambio di diversi tipi di dati e comandi tra client e server.

3 Come compilare ed eseguire il gioco Wordle

3.1 Ambiente di test

Il progetto è stato testato sui seguenti sistemi operativi :

- Linux Ubuntu 18.04.6 LTS con Java 11
- Windows 11 con Java 14

3.2 Istruzioni per l'Installazione e l'Esecuzione del Gioco

Per eseguire il gioco ci sono due modi :

- **Da riga di comando** [Passi da seguire per compilare ed eseguire il gioco] :
  1. Entrare nella cartella **src**.
  2. Una volta nella cartella src, aprire tre (o più) terminali. Di questi un terminale è dedicato all'avvio del ***server***. E due (o più) sono dedicati al ***client***, per effettuare vari test di concorrenza e di condivisione del risultato di una partita.
  3. Sul terminale dedicato al SERVER :
     - ***Per compilare*** :
       `javac -d ./../out -cp ".:gson-2.8.2.jar" WordleServerMain.java` (su Linux)
       `javac -d ./../out -cp ".;gson-2.8.2.jar" WordleServerMain.java` (su Windows)
     - ***Per eseguire*** :
       `java -cp "./../out/:gson-2.8.2.jar" WordleServerMain` (su Linux)
       `java -cp "./../out/;gson-2.8.2.jar" WordleServerMain` (su Windows)
  4. Sul terminale dedicato al CLIENT :
     - ***Per compilare*** :
       `javac -encoding UTF-8 -d ./../out WordleClientMain.java` (su Windows)
       `javac -d ./../out WordleClientMain.java` (su Linux)
     - ***Per eseguire*** :
       `java -cp ./../out WordleClientMain` (su Linux/windows)
- **Con i file eseguibili JAR** (da riga di comando) [Passi per eseguire il gioco] :
  1. Sul terminale dedicato al SERVER, eseguire :
     `java -jar server.jar`
  2. Sul terminale dedicato al CLIENT, eseguire :
     `java -jar client.jar`