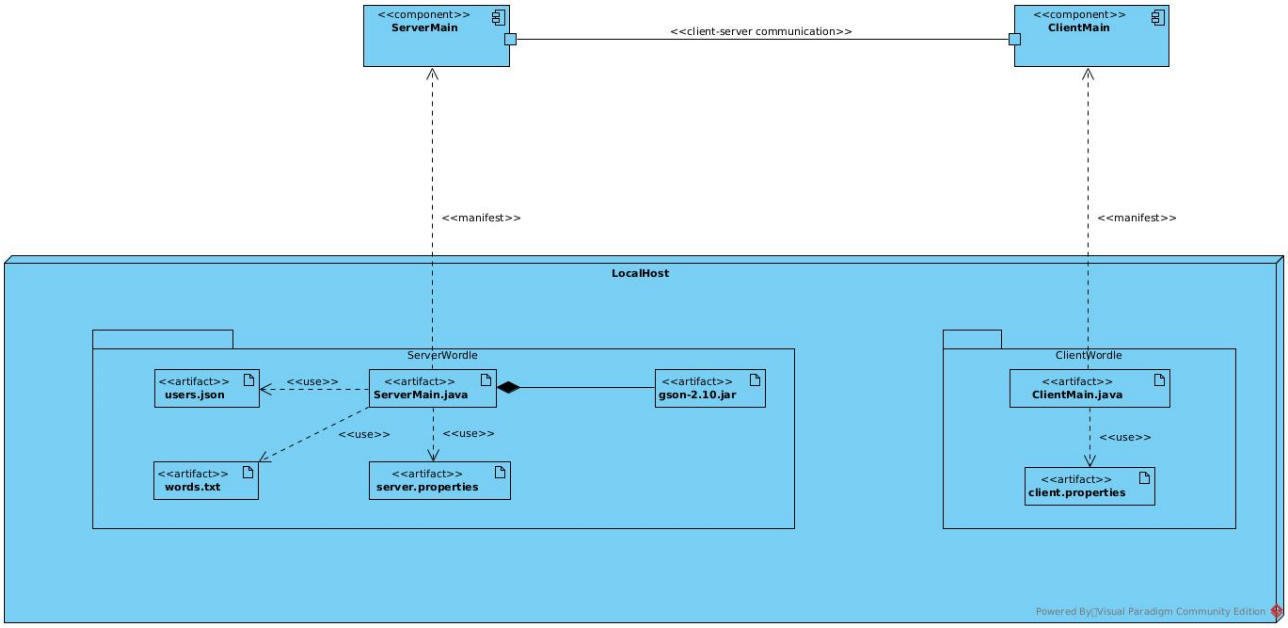
Nome: Davide
Cognome: Orsucci
Matricola: 616802

# Laboratorio di Reti WORDLE: un gioco di parole 3.0

# Progetto di Fine Corso A.A. 2022/23

## *Visione generale*

Il progetto è diviso in due parti principali: WordleServer e WordleClient, due directory contenenti rispettivamente il codice sorgente per l'applicazione lato Server e lato Client di WORDLE. Di seguito una vista di dislocazione che mostra le principali componenti del progetto e le loro dipendenze.




All'interno di WordleClient si trova rispettivamente:

- ClientMain.java : file contenente il codice sorgente del Client.
- client.properties : file contenente i parametri per configurare il corretto funzionamento di ClientMain.java, tra cui l'hostname (localhost) e il numero di porta a cui deve collegarsi per accedere al servizio.

All'interno di WordleServer si trova:

- ServerMain.java : file contenente il file sorgente del Server
- users.json : file JSON contenente i dati degli utenti su cui il Server preleva e deposita i dati degli utenti.
- server.properties : file contenente i parametri per configurare il corretto funzionamento del ServerMain.java, tra cui numero di porta e indirizzo di multicast utilizzato per spedire le notifiche degli utenti, il numero di porta a cui i processi Client devono collegarsi, il nome del file JSON contenenti i dati degli utenti per mantenere lo stato del sistema persistente.
- words.txt : vocabolario utilizzato per estrarre la parola segreta.
- gson-2.10.jar : libreria utilizzata per serializzare/deserializzare i dati presenti su users.json.

entrambe le cartelle contengono anche gli artefatti con estensione .jar, non mostrate in figura.

Nome: Davide
Cognome: Orsucci
Matricola: 616802

## *CLIENT*

Dal processo ClientMain vengono creati due Threads:

- main: gestisce la logica del gioco lato client. Si occupa principalmente di inviare al Server i comandi ottenuti in input da tastiera dal giocatore attraverso una connessione TCP e di mostrare all'utente l'esito mediante il metodo checkResult( ).
- sharingReceiver: aggiunge il client a un gruppo di multicast e raccoglie i datagrammi UDP contenenti i risultati condivisi dagli altri utenti. Le notifiche raccolte vengono mostrate quando l'utente sceglie l'opzione show me sharing.

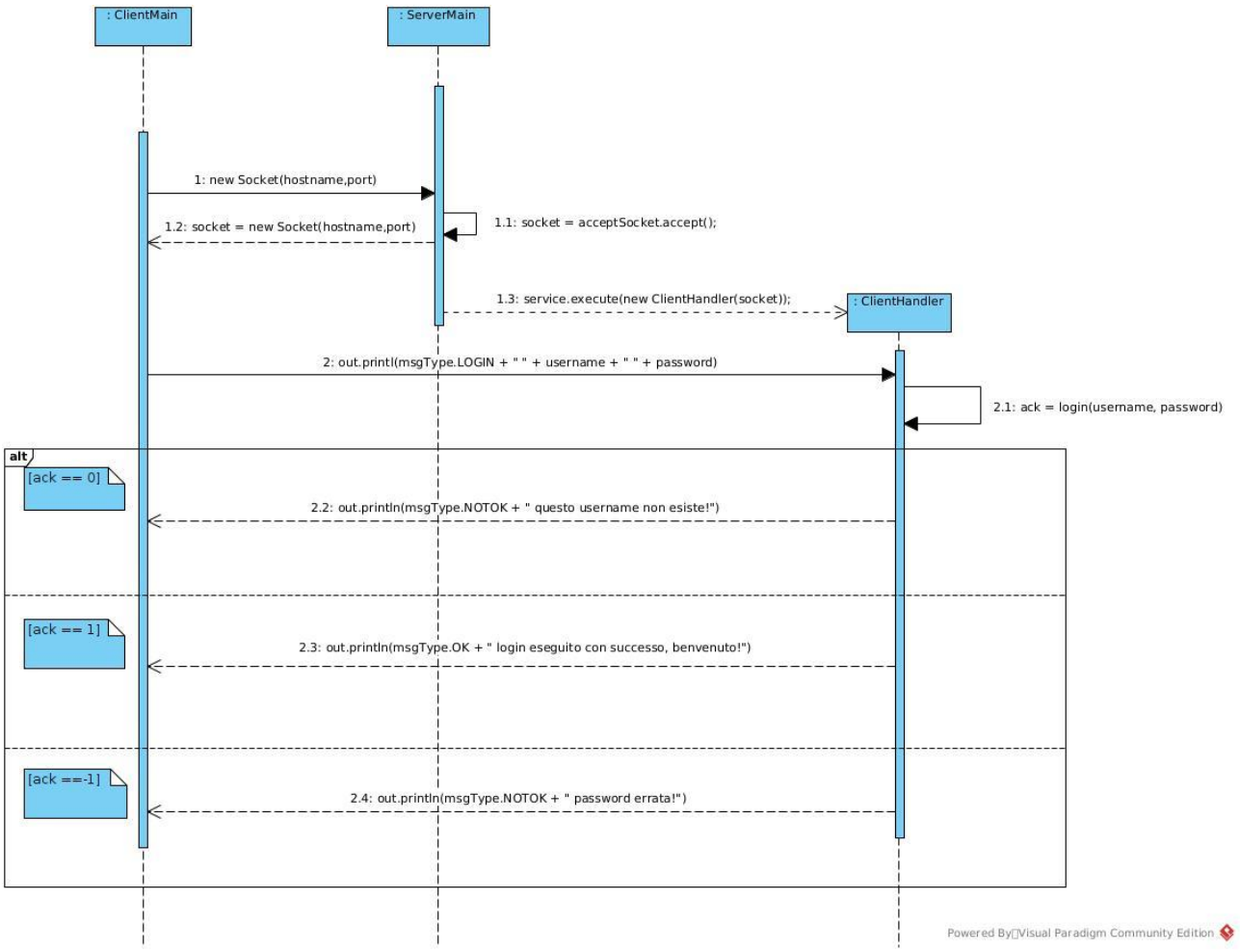
Client e Server comunicano mediante lo scambio di messaggi su una connessione TCP. Per facilitare la comunicazione tra Client e Server ho definito un protocollo di comunicazione tra Client e Server che prevede lo scambio di stringhe con il seguente formato standard:

String messaggio = msgType.TIPOMESSAGGIO + " " + parametro1 + " " +…+parametroN

- msgType.TIPOMESSAGGIO indica il tipo di messaggio di richiesta/risposta
- parametroI indica eventuali parametri che sono necessari al Server per la eseguire richiesta

tutti i campi sono separati da uno spazio.

Di seguito un diagramma di sequenza che mostra come avviene lo scambio di messaggi tra ClientMain e ServerMain per instaurare una connessione e successivamente eseguire una richiesta di login:



Nome: Davide
Cognome: Orsucci
Matricola: 616802
Il messaggio spedito dall'applicazione Client ha la seguente forma:

msgType.LOGIN username password

Il server dopo aver ricevuto la richiesta risponde al Client con un messaggio di riscontro. Il client utilizzando il metodo checkResult( ) controlla il formato del messaggio, che ha la seguente forma

msgType.OK messaggioRisposta

msgType.NOTOK messaggioErrore

- msgType.OK → la richiesta è andata a buon fine, di seguito al primo campo sono riportati i risultati da stampare a schermo dell'operazione richiesta.
- msgTyoe.NOTOK → la richiesta non è andata a buon fine, di seguito al primo campo viene riportato un messaggio di errore che deve essere mostrato dal Client a schermo.

Lato Client come detto prima sono in esecuzione due thread, main e sharingReceiver.

Entrambi i threads accedono alla struttura ConcurrentLinkedQueue<String> notifies in modo concorrente: sharingReceiver vi accede per modificarla e aggiungere le notifiche ricevute attraverso datagrammi UDP, mentre main vi accede per leggere le notifiche ricevute. La coda è thread safe, perciò in ambiente concorrente dove c'è contesa per le risorse l'accesso avviene in maniera sicura.

## *SERVER*

All'avvio del Server viene deserializzato il file users.json contentente i dati degli utenti registrati al gioco. Questi dati vengono deserializzati in una ConcurrentHashMap<String, User> per permettere l'accesso concorrente ai dati degli utenti in modo Thread safe permettendo inoltre di accedere con pochi accessi in memoria i dati degli utenti attraverso una chiave univoca rappresentata dallo username dell'utente.

Per deserializzare i dati degli utenti ho utilizzato la libreria Gson e ho definito una classe di appoggio User contenente i dati che devono essere preservati di ciascun utente tra cui username, password, statistiche utente, un valore booleano che indica se ha vinto la partita giornaliera oppure no, se ha vinto l'ultima partita e quanti tentativi gli rimangono per la corrente partita.

Ogni qualvolta viene effettuata una modifica a uno degli attributi dell'utente sopracitati viene chiamato il metodo updateUsers( ).

Il metodo synchronized updateUsers( ) permette di aggiornare il file users.json con le modifiche apportate a ConcurrentHashMap<> users. Il metodo risponde a 3 esigenze:

- Consistenza, evitare che ci siano problemi di coerenza tra il file users.json e la ConcurrentHashMap.
- Persistenza, necessità di mantenere i dati nella base di dati per poter mantenere lo stato del sistema.
- Serializzazione, la modifica al file viene effettuata da un ClientHandler alla volta in maniera sicura.

Il Server è stato realizzato mediante CachedThreadPool: quando viene instaurata una nuova connessione da parte di un processo Client e è stata quindi creata una Socket per far comunicare i due processi, la CachedThreadPool crea un nuovo Thread di tipo ClientHandler(Socket socket) che si occupa di soddisfare le richieste del Client finché non decide di uscire.

Sempre all'avvio del server viene avviato il thread WordExtractor che si occupa di svolgere azioni di routine per il corretto funzionamento della logica del gioco, in particolare si occupa di preparare una nuova partita mediante le seguenti azioni:

Nome: Davide
Cognome: Orsucci
Matricola: 616802

- estrae periodicamente una nuova parola da indovinare chiamando il metodo extractSecretWord() il quale sceglie in modo casuale una delle parole presenti all'interno del file words.txt mediante RandomAccessFile. RandomAccessFile permette di leggere le parole al suo interno attraverso un file pointer senza dover accedere sequenzialmente le righe del file.
- si occupa di ripristinare lo status di vittoria quando viene estratta una nuova parola.
- ripristina i consigli ricevuti dagli utenti.

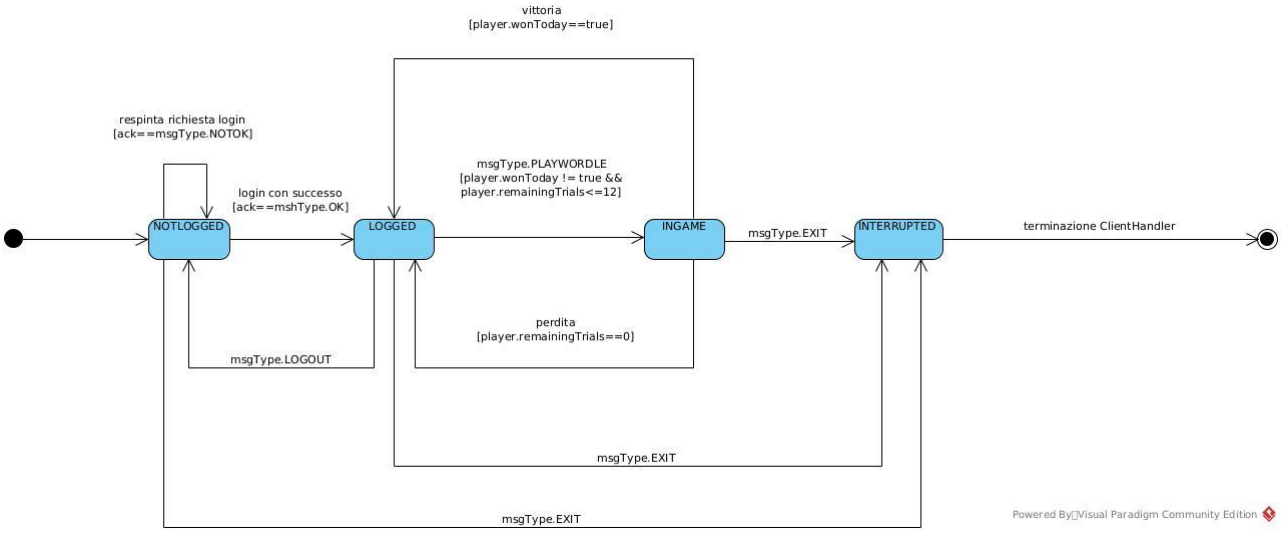
Ciascun ClientHandler controlla i messaggi ricevuti dal Client e interpreta i messaggi ricevuti facendo un pattern matching:

Il server riceve un messaggio da parte del Client, estrae da esso il campo di tipo msgType attraverso la quale il Server controlla il tipo di richiesta da soddisfare e in base a essa estrae i campi successivi nel messaggio necessari per performare la richiesta. Dopo aver elaborato la richiesta il server invia un messaggio di riscontro al cui inizio c'è msgType.OK se la richiesta è stata soddisfatta con successo e msgType.NOTOK se non è stato possibile performarla.

ClientHandler tiene traccia dello stato della partita usando 4 stati differenti:

- NOTLOGGED (quando player == null): l'utente si è connesso a server ma non ha effettuato il login.
- LOGGED (quando player != null): l'utente ha effettuato il login correttamente con msgType.LOGIN.
- INGAME: l'utente ha fatto richiesta per giocare attraverso msgType.PLAYWORDLE. Il giocatore può passare allo stato di INGAME se è nello stato di LOGGED, non ha ancora finito i tentativi giornalieri e non ha ancora vinto la partita.
- INTERRUPTED: il giocatore passa allo stato INTERRUPTED quando invia un messaggio msgType.EXIT. Alla ricezione di tale segnale il thread ClientHandler termina.

Il grafo sottostante rappresenta la transizione tra gli stati per un utente attraverso una final state machine




Un giocatore può iniziare a mandare messaggi di tipo msgType.SENDWORD soltanto quando si trova nello stato INGAME.

Quando un giocatore invia una richiesta di inviare una parola col seguente formato

msgType.SENDWORD guessedWord

il ClientHandler si occupa di controllare:

Nome: Davide
Cognome: Orsucci
Matricola: 616802

- se guessedWord.equals(secretWord), in tal caso il giocatore ha vinto la partita giornaliera, vengono modificate le sue statistiche di gioco. Viene aggiunto a wordSuggestions il risultato "++++++++++".
- Se checkVocabulary(guessedWord), significa che la guessedWord inviata dal giocatore è presente nel vocabolario, viene decrementato il numero di tentativi rimanenti al giocatore. La ricerca viene effettuata col metodo boolean checkVocabulary(String word) che accede al file words.txt mediante RandomAccessFile e effettua una ricerca binaria di costo logaritmico nel numero delle righe del file words.txt. In caso affermativo, viene preparato un suggerimento al Client formato da +,?,x. Questo suggerimento viene aggiunto a ArrayList<String> wordSuggestions.
- Se la guessedWord spedita dal Client non è presente nel vocabolario allora viene mandato un messaggio di errore al client avente prefisso msgType.NOTOK.

Al termine della procedura viene spedito al client un messaggio con l'esito dell'operazione (prefisso msgType.OK e msgType.NOTOK).

Ciascun ClientHandler tiene una propria ArrayList<String> wordSuggestions. Questa viene utilizzata per la condivisione dei propri risultati agli altri giocatori attraverso il metodo

sendUDPMessage(String message, String ipAddress, int port)

Dove ipAddress e port sono rispettivamente l'InetAddress e il numero di porta utilizzati per creare il DatagramPacket per spedire i risultati nel gruppo di multicast alla quale partecipano tutti i Client che hanno effettuato la connessione al server.

Alla pressione della combinazione di tasti CTRL+C (invio di segnale SIGINT), viene eseguito uno Shutdown Hook, che permette di eseguire un frammento di codice prima che la JVM termini bruscamente.

Lo Shutdown Hook esegue tre operazioni principalmente:

1. Riporta sul file users.json tutte le modifiche effettuate in ConcurrentHashMap<String, User> users nel caso in cui ce ne siano alcune che non sono ancora state riportate al momento della chiamata allo Shutdown Hook.
2. Esegue l'operazione service.shutdown( ) attraverso la quale la CachedThreadPool smette di accettare nuove richieste di Client da Gestire.
3. Attende la terminazione dei threads non ancora terminati della CachedThreadPool service con un timeout di 4 secondi. Nel caso in cui i threads non siano stati terminati entro il tempo specificato, viene mostrato un messaggio sul server.

Nome: Davide
Cognome: Orsucci
Matricola: 616802

# *Manuale di istruzioni*

Per compilare il Server di WORDLE è sufficiente navigare le cartelle del progetto, entrare nella cartella WordleServer e da terminale digitare il seguente comando:

```
javac -cp gson-2.10.jar ServerMain.java
```

Per mandare in esecuzione il server digitare:

```
java -cp gson-2.10.jar ServerMain
```

Per compilare il Client navigare le cartelle, entrare nella cartella WordleClient e da terminale digitare il seguente comando:

```
javac ClientMain.java
```

Per eseguire il programma client digitare:

```
java ClientMain
```

al suo avvio ClientMain tenterà di connettersi di connettersi a ServerMain, è importante che venga mandato in esecuzione prima ServerMain e successivamente ClientMain altrimenti verrà mostrato un messaggio di connessione rifiutata da parte del Client.