Wordle 3.0

***Matteo Giorgi* 517183**

Il progetto consiste nella implementazione di Wordle, un gioco di parole web-based sviluppato da Josh Wardle nel 2021, acquistato poi dal New York Times a fine 2022.

Ogni 24h il gioco estrae casualmente dal proprio dizionario una Secret-Word di 5 lettere che il giocatore deve indovinare proponendo una Guessed-Word per ciascuno dei 6 tentativi massimi consentiti. Ad ogni tentativo, Wordle risponderà con indizi utili riguardo le lettere che compongono la Guessed-Word così da aiutare il giocatore a indovinare la Secret-Word giornaliera.

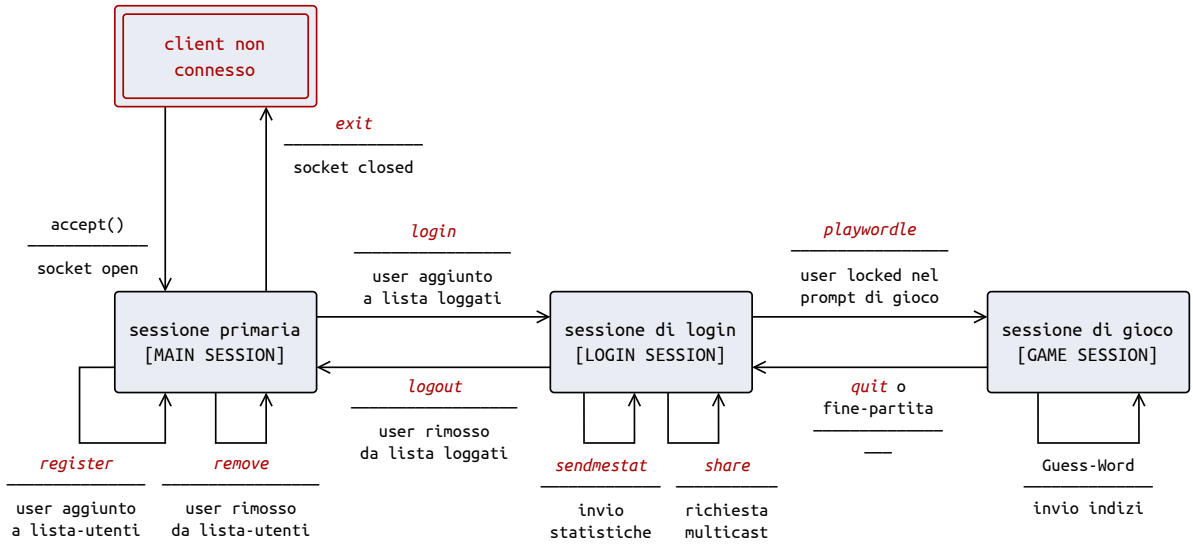
Questa implementazione consiste in una versione semplificata del gioco, che conserva la logica di base dell'originale ma apporta modifiche su alcune funzionalità come la condivisione social dei risultati (realizzata qui con un gruppo multicast), e l'assenza di una interfaccia grafica (sostituita da una semplice Command-Line UI).

La presente relazione è affiancata da documentazione JavaDoc secondo quanto specificato nelle Tecnical-Resources Oracle. ***Questo testo esiste anche in html wiki-page.***

Wordle 3.0 usa una classica struttura client-server. Il server legge il proprio file di configurazione e si occupa di caricare in memoria l'elenco degli utenti, l'elenco delle parole (dizionario) e rimanere in attesa di connessioni su una welcome-socket (`ServerSocket`) appositamente allocata su una porta predefinita nel file di configurazione. Agganciato un client, il server lancierà dunque un nuovo Runnable (`Game`) con cui verranno soddisfatte le richieste, per poi rimettersi in attesa di una nuova connessione. Il client invece, dopo la lettura del proprio file di configurazione, ha l'unico scopo di connettersi al server con una socket (`Socket`) e inviare comandi sottoforma di *lines* (stringhe terminanti con il carattere di line-break).

Struttura del Progetto

Prima di entrare nelle specifiche dell'implementazione ecco qua sotto l'ASF che illustra i possibili stati di un client nelle varie fasi di gioco (si consideri ovviamente che client e server abbiano superato la fase iniziale di setup).



Modelli principali

- `Word` e `WordList` gestiscono le parole del gioco: la classe `Word` rappresenta una singola parola, mentre `WordList` il dizionario usato da *Wordle* per estrarre ciclicamente la *Secret-Word* e controllare la validità delle *Guessed-Words* inserite dall'utente in gioco.
- `User` e `UserList` in modo analogo alle classi precedenti, rappresentano rispettivamente il singolo utente e l'insieme degli utenti registrati sul server del gioco.

Funzionalità del server

- `ServerSetup` e `ServerMain` sono le classi che descrivono le proprietà e identificano il punto di ingresso principale del server.
- `Game` è la classe responsabile della sessione di gioco per ciascun client, secondo i quattro stati specificati nell'ASF.
- `MulticastSender` gestisce la condivisione dei risultati attraverso un gruppo multicast.

Funzionalità del client

- `ClientSetup` e `ClientMain` analogamente alle loro controparti, rappresentano le proprietà e il punto di ingresso del client.
- `MulticastReceiver` gestisce la condivisione dei risultati attraverso un gruppo multicast.

Classe `Word`

Classe che rappresenta la parola che i giocatori devono indovinare, è implementata usando due variabili private.

- *`currentWord`*: `String` che identifica la *Secret-Word* corrente.
- *`userSet`*: `Set<String>` che contiene i nomi degli utenti che hanno già giocato la *Secret-Word* corrente. Un utente che avesse già giocato la *Secret-Word* corrente e chiedesse di iniziare una nuova partita, rimarrebbe in `[LOGIN SESSION]` (si veda AST).

Il costruttore della classe prende come parametro una `String` che rappresenta la parola da indovinare; non è consentito creare una parola *null*, causerebbe il lancio di una `IllegalArgumentException`.

Oltre ai metodi *`getWord`*, *`containsUser`* e *`addUser`*, la classe contiene anche *`getMask`*, necessario a `Game` per fornire informazioni al client sulla correttezza della *Guessed-Word* inserita, sottoforma di una maschera di caratteri speciali (`X`, `+`, `?`) come da specifica.

Classe `WordList`

Classe che rappresenta il vocabolario utilizzato da *Wordle* per estrarre la *Secret-Word* e controllare la validità delle *Guessed-Word* inserite dagli utenti durante una partita. Di seguito la struttura base.

- *`wordVocabulary`*: `List<String>` che contiene le parole del vocabolario.
- *`currentWord`*: istanza della classe `Word` che rappresenta la parola attualmente selezionata come *Secret-Word*.
- *`wordExtractor`*: `ScheduledExecutorService` che estrae una parola casuale dal vocabolario a intervalli regolari di tempo.

La classe contiene anche il `Runnable` *`extractWord`* che rappresenta il task di estrazione casuale di una nuova *Secret-Word* dal vocabolario, eseguito da `wordExtractor` ogni `wordTimer` secondi (tempo specificato nel file

di configurazione del server).

Classe `User`

Classe che rappresenta l'utente come una mappa chiave-valore che ne specifica le proprietà.

- *`user`*: nome dell'utente.
- *`password`*: password per il login dell'utente.
- *`giocate`*: numero di partite giocate dall'utente.
- *`vinte`*: numero di partite vinte dall'utente.
- *`streaklast`*: lunghezza della serie di vittorie più recente dell'utente.
- *`streakmax`*: lunghezza massima della serie di vittorie ottenuta dall'utente.
- *`guessd`*: distribuzione che indica i tentativi impiegati dall'utente per arrivare alla soluzione nelle partite giocate.

Per evitare di esporre i metodi di modifica della mappa, invece di estendere una delle implementazioni di `Map`, è stata usata la variabile privata *`user`* di tipo `Map<String, Object>` per identificare l'utente con le sue proprietà.

Questa scelta permette di avere una struttura sicura e flessibile, facilmente modificabile al termine di ogni partita con l'apposito metodo *`update`*, con il quale `Game` può aggiornare i dati dell'utente.

Oltre ai metodi getter con i quali recuperare i sette valori della mappa, la classe contiene il metodo *`copy`* che permette di creare una copia profonda dell'utente.

La classe ha un unico costruttore che permette di creare un nuovo utente partendo da una `Map<String, Object>`. Non è consentito creare un oggetto della classe `User` usando una mappa *null*, incompleta o contenente associazioni chiave-valore di tipo sbagliato: questi casi causerebbero il lancio di una `NullPointerException` o di una `IllegalArgumentException`.

Classe `UserList`

Il file `UserList.java` definisce la classe `UserList`, che rappresenta l'elenco degli utenti del gioco Wordle. Ecco una panoramica iniziale del suo contenuto:

La classe `UserList` implementa l'elenco degli utenti. Ha due strutture dati principali:

- **userRegistrati**: Una `Map<String, User>` che contiene gli utenti registrati al gioco. La chiave è il nome dell'utente e il valore è un oggetto `User` che rappresenta l'utente.
- **userLoggati**: Un `Set<String>` che contiene i nomi degli utenti attualmente loggati al gioco.

La classe utilizza le librerie `com.google.gson.*` per gestire la serializzazione e deserializzazione degli utenti in formato JSON, permettendo di salvare e caricare l'elenco degli utenti da e verso un file.

La classe `UserList` fornisce metodi per aggiungere, rimuovere, recuperare e manipolare gli utenti. Inoltre, gestisce le operazioni di login e logout, garantendo che un utente possa essere loggato una sola volta.

Questa classe è fondamentale per la gestione degli utenti nel gioco, memorizzando le informazioni di ogni utente e garantendo l'integrità dei dati.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `ServerSetup`

Questa classe estende `Properties` e si preoccupa di leggere il file di configurazione del server all'avvio, memorizzando le varie proprietà nelle appropriate strutture dati. Le proprietà disponibili sono:

1. **PORT**: Il numero di porta su cui il server si mette in ascolto. È un valore intero.
2. **PATH_VOCABULARY**: Il percorso del file che contiene il vocabolario (dizionario) del gioco. È una stringa.
3. **PATH_JSON**: Il percorso del file JSON che contiene i dati degli utenti. È una stringa.
4. **WORD_TIMER**: Il tempo che intercorre tra le pubblicazioni della parola segreta. È un valore intero.
5. **MULTICAST_GROUP_ADDRESS**: L'indirizzo del gruppo multicast, utilizzato per la condivisione dei risultati. È una stringa.

La classe `ServerSetup` quindi fornisce al server tutte le informazioni necessarie per la sua configurazione iniziale, permettendo di personalizzare vari aspetti del gioco e della sua esecuzione.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `ServerMain`

Il file `ServerMain.java` definisce la classe `ServerMain`, che sembra essere la classe principale del server Wordle. Ecco una descrizione iniziale del suo contenuto:

Questa classe contiene il metodo `main` ed è responsabile per l'inizializzazione e l'esecuzione del server. Ha i seguenti compiti principali:

1. **Creazione delle Strutture Dati**: All'avvio, il server crea le strutture dati necessarie, come l'elenco degli utenti (`UserList`) e il dizionario (`WordList`).
2. **Attesa di Connessioni**: Il server si mette in attesa di connessioni dai client. Una volta ricevuta una connessione, gestisce la comunicazione con il client.
3. **Gestione di Socket e Thread Pools**: La classe si occupa anche di gestire la chiusura delle socket e dei thread pools.

In termini di variabili, vediamo:

- **PATH_CONF**: Rappresenta il percorso del file di configurazione del server.
- **serverProperties**: Un'istanza di `ServerSetup` che memorizza le proprietà del server.
- **listaUtenti**: Un'istanza di `UserList` che memorizza gli utenti registrati al gioco.
- **listaParole**: Un'istanza di `WordList` che memorizza le parole del gioco.
- **welcomeSocket**: Una `ServerSocket` utilizzata per accettare connessioni dai client.

La classe `ServerMain` funge da punto di ingresso per il server, gestendo l'inizializzazione, l'attesa di connessioni e la comunicazione con i client.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `Game`

Il file `Game.java` definisce la classe `Game`, che rappresenta un thread per gestire l'interazione tra il server e un client. Ecco una panoramica iniziale del suo contenuto:

Questa classe implementa un thread che propone una interfaccia testuale al client, suddivisa in diverse fasi:

1. [**MAIN SESSION**]: È la fase iniziale in cui l'utente ha diverse opzioni:
    - **exit**: Uscire dal programma.
    - **register**: Registrarsi al gioco.
    - **remove**: Cancellarsi dal gioco.

    - **login**: Autenticarsi al gioco.
2. [**LOGIN SESSION**]: Una volta autenticato, l'utente ha altre opzioni:
    - **playwordle**: Iniziare una nuova partita.
    - **sendmestat**: Richiedere le proprie statistiche di gioco.
    - **sharemestat**: Condividere le proprie statistiche.

La classe `Game` quindi gestisce l'interazione con l'utente attraverso una serie di comandi testuali, guidandolo attraverso le diverse fasi del gioco e fornendo una serie di funzionalità basate sullo stato di sessione dell'utente.

Dalla panoramica iniziale, sembra che questa classe abbia un ruolo centrale nell'interazione con l'utente e nella logica del gioco.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `MulticastSender`

Il file `MulticastSender.java` definisce la classe `MulticastSender`, che sembra essere responsabile dell'invio di notifiche su un gruppo multicast. Ecco una breve descrizione del suo contenuto:

La classe `MulticastSender` implementa l'interfaccia `Runnable` e gestisce l'invio di notifiche tramite un gruppo multicast:

- **multicastGroupPort**: Un numero intero che rappresenta la porta del gruppo multicast.
- **multicastGroupAddress**: Una stringa che rappresenta l'indirizzo del gruppo multicast.
- **queue**: Una coda utilizzata per conservare temporaneamente le notifiche inviate dal server. Queste notifiche sono in attesa di essere lette e inviate sul canale multicast.

Dalla descrizione, sembra che `MulticastSender` funzioni come un thread che rimane in attesa (probabilmente utilizzando `wait()`) fino a quando non viene notificato dell'aggiunta di una nuova notifica. Una volta ricevuta questa notifica, la legge dalla coda e la invia sul gruppo multicast.

Questa classe svolge un ruolo chiave nella funzionalità di condivisione dei risultati del gioco, permettendo ai client di ricevere notifiche in tempo reale tramite il gruppo multicast.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `ClientSetup`

Il file `ClientSetup.java` definisce la classe `ClientSetup`, che rappresenta le proprietà del client. Ecco una breve descrizione del suo contenuto:

Questa classe estende `Properties` e ha il compito di leggere il file di configurazione del client all'avvio, memorizzando le varie proprietà nelle appropriate strutture dati. Le proprietà disponibili sono:

1. **HOSTNAME**: Il nome del server al quale il client si connetterà. È una stringa.
2. **PORT**: Il numero di porta del server. È un valore intero.
3. **MULTICAST_GROUP_ADDRESS**: L'indirizzo del gruppo multicast, utilizzato per la condivisione dei risultati. È una stringa.
4. **MULTICAST_GROUP_PORT**: Il numero di porta del gruppo multicast. È un valore intero.

Similmente a `ServerSetup`, la classe `ClientSetup` fornisce al client tutte le informazioni necessarie per la sua configurazione iniziale, permettendo di personalizzare vari aspetti della connessione e dell'interazione con il server.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `ClientMain`

Il file `ClientMain.java` definisce la classe `ClientMain`, che rappresenta il punto di ingresso principale per il client. Ecco una breve descrizione del suo contenuto:

Questa è la classe principale che rappresenta il client:

- **PATH_CONF**: Contiene il percorso del file di configurazione del client.
- **clientProperties**: Un'istanza della classe `ClientSetup` che memorizza le proprietà del client.
- **multicastListener**: Un thread che rimane in ascolto delle notifiche sul gruppo multicast.
- **multicastReceiver**: Un'istanza della classe `MulticastReceiver` che memorizza le notifiche ricevute.
- **logStatus**: Indica lo stato del log (false = non loggato, true = loggato).

La classe `ClientMain` inizializza le configurazioni necessarie per il client, si connette al server e gestisce l'interazione con l'utente. Utilizza le configurazioni fornite dalla classe `ClientSetup` e le notifiche multicast attraverso la classe `MulticastReceiver`.

Questa classe è essenziale per avviare e gestire il comportamento del client nel gioco Wordle.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?

Classe `MulticastReceiver`

Il file `MulticastReceiver.java` definisce la classe `MulticastReceiver`, che gestisce la ricezione di notifiche da un gruppo multicast. Ecco una breve descrizione del suo contenuto:

La classe `MulticastReceiver` estende `ConcurrentLinkedQueue<String>` e implementa l'interfaccia `Runnable`. Questo suggerisce che la classe funge da coda thread-safe di stringhe (probabilmente le notifiche ricevute) e può essere eseguita come un thread.

Ecco le principali variabili e funzionalità della classe:

- **multicastGroupPort**: Il numero di porta del gruppo multicast.
- **multicastGroupAddress**: L'indirizzo del gruppo multicast.
- **userName**: Il nome dell'utente che sta ascoltando le notifiche sul gruppo multicast.

La classe `MulticastReceiver` rimane in ascolto delle notifiche sul gruppo multicast e aggiunge le notifiche ricevute alla coda delle notifiche. Questo permette ai client di ricevere notifiche in tempo reale e di processarle in modo asincrono.

Questa classe, insieme a `MulticastSender`, consente una comunicazione multicast tra il server e i client, fornendo una funzionalità di condivisione in tempo reale tra i giocatori.

Proseguiamo con un'analisi più dettagliata di questa classe o preferisci passare alla successiva?