Relazione “Wordle 3.0” - Laboratorio 3 - Francesco Dore - A.A. 22/23

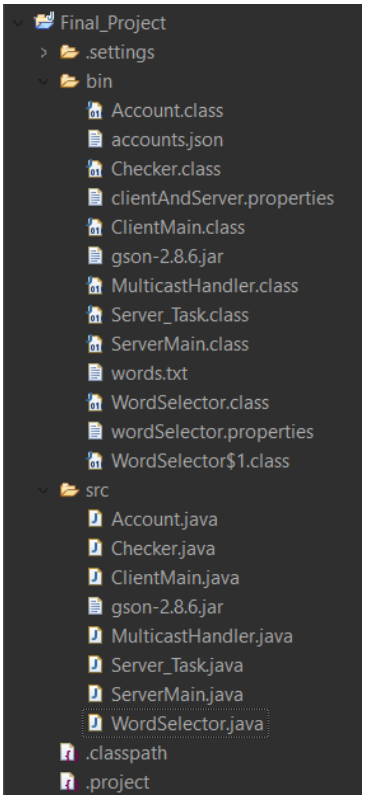
Introduzione

“Wordle 3.0: un gioco di parole” è una versione semplificata del famoso “Wordle”. All’interno del gioco, l’obiettivo è quello di indovinare la “parola del giorno” usando parole esistenti nel vocabolario di riferimento che siano della stessa lunghezza e cercando di ottenere più informazioni possibili riguardo la parola nascosta ottenendo suggerimenti dalle varie parole scelte. In particolar modo, se una lettera si trova al posto giusto o una lettera è compresa nella parola ma in una posizione diversa, verrà segnalato all’utente.
Alla fine di una partita, l’utente può condividere i propri risultati e vedere le statistiche degli amici sui vari social associati.

Struttura del progetto

Il progetto è una versione semplificata e diversa da quello che è il vero fenomeno internazionale. Le parole sono lunghe 10 caratteri, e l’utente ha 12 tentativi per cercare di indovinarla. Come nel gioco originale, è possibile inoltre vedere i risultati di chi condivide la propria partita, ma non su una vera social network, bensì su un gruppo di multicast, del quale fanno parte tutti i client che hanno effettuato il login. Tutte le informazioni sullo stato della partita, sulle statistiche degli altri giocatori e su sè stessi e l’interazione stessa con l’applicazione avvengono mediante interfaccia da linea di comando (CLI). Il progetto è stato interamente sviluppato su Eclipse ed è sviluppato secondo il paradigma Client-Server: ogni utente è rappresentato da un Client, che può collegarsi al Server per accedere e giocare. Il Client in quanto tale manda delle richieste, e ottiene delle risposte dal Server, che gli permettono di mostrare a video

all’utente informazioni riguardanti il suo stato e la sua partita. L’utente è inoltre capace di vedere le statistiche degli altri utenti, in quanto dopo il login l’utente si unisce ad un gruppo di Multicast, sul quale possono essere condivisi i vari esiti delle partite e visualizzate le statistiche di tutti coloro che condividono i propri risultati.
Nella creazione di questo progetto, il Server è da intendersi sempre attivo. Di fatto, la connessione dei Client può avvenire solo se quest’ultimo è già “acceso”, e il riavvio del server comporta la scelta di una nuova parola del giorno, pur garantendo la permanenza dei dati.
Il progetto è fisicamente strutturato come segue:



Esistono due cartelle, src e bin:

- Src contiene tutte le classi del progetto (che sono “Account.java”, “Checker.java”, “ClientMain.java”, “MulticastHandler.java”, “Server_Task.java”, “ServerMain. java” e “WordSelector.java”, 7 classi in totale);
- Bin, che contiene i file di configurazione, i file .class ottenuti dopo la compilazione e un ulteriore file chiamato “gson-2.8.6.jar”, una libreria esterna.

Si noti la presenza del file ”gson-2.8.6.jar” in entrambe le cartelle bin e src. Si tratta di una libreria che ci permette di lavorare in maniera semplice coi file JSON.
Il file si trova in entrambe le cartelle per questioni di praticità, ma la scelta migliore, per ordine e pulizia dei file, sarebbe quella di avere il file in una cartella separata chiamata “lib” che contiene tutte le librerie esterne utilizzate. Eclipse non ha generato automaticamente questa cartella, ed è stato più comodo avere il file .jar in bin e src, ma generalmente, la procedura da seguire è quella appena descritta.

Analizziamo singolarmente i file contenuti nelle cartelle:

Contenuto di Src

ServerMain.java

E’ il file che contiene il metodo main del server. Consta di poche righe di codice, in quanto la maggior parte del lavoro viene svolta dal ThreadPool creato al suo interno. Il server accede ad un file contenuto in bin “server.properties” che contiene tutti i parametri di configurazione necessari alla compilazione e all’avvio corretti.

Server_Task.java

E’ il file che contiene la classe utilizzata come task per il thread pool presente in “ServerMain.java”. Svolge la maggior parte del lavoro, ed è il vero intermediario tra l’utente e i suoi dati, in quanto è colui che risponde alle richieste del Client associatogli.

ClientMain.java

E’ il file che viene “utilizzato” dai vari utenti per accedere al gioco. Permette di interfacciarsi col Server, chiedendo informazioni e, quando ricevute, mostrandole sulla riga di comando all’utente.
Per un corretto avvio e connessione al server, legge dal file “client.properties” (contenuto in bin) i parametri necessari.

WordSelector.java

E’ la classe atta alla selezione della parola giornaliera. La parola viene scelta da un file di parole preimpostato, che viene specificata in un file di configurazione esterno “wordSelector.properties”.
Anche l’intervallo di scelta della nuova parola viene specificato nello stesso file di configurazione.

Checker.java

E’ una classe il cui scopo è quello di mantenere una lista dei giocatori online. I suoi metodi sono pubblici e permettono al Server di aggiornare la coda ogniqualvolta un Client effettui il login o il logout, permettendo rispettivamente l’aggiunta o la cancellazione dalla lista.

Account.java

Definisce la struttura di un oggetto di tipo Account. E’ necessaria per la lettura e scrittura mediante JsonReader, classe della libreria esterna GSON utilizzata per

aggiornare il file JSON di tutti gli utenti.

MulticastHandler.java

All'avvio del Client, dopo la fase di login, viene lanciato un thread esterno, appartenente a questa classe. Il thread rimane in ascolto di eventuali messaggi scambiati sul gruppo di Multicast, e permette di visualizzare tutti i messaggi ricevuti dall'avvio del thread mediante un metodo "showMeSharing()".

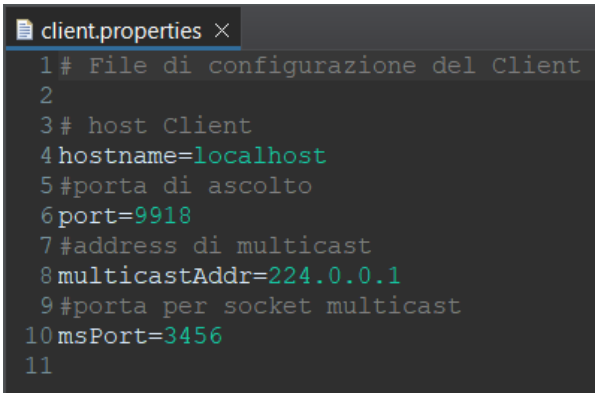
Contenuto di Bin

"<nome>.class"

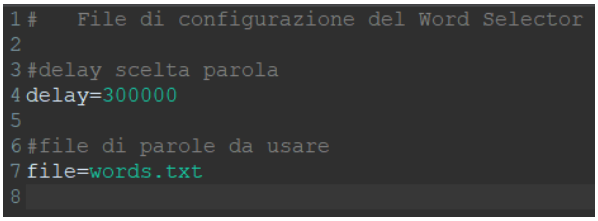
Sono i file .class ottenuti dopo aver compilato i file .java
Niente in più da aggiungere in quanto già detto nella sezione precedente.

"<nome>.properties"

Sono i file di configurazione per le diverse classi: uno per il Client, uno per il Server, uno per il Selettore di parole (wordSelector).



client.properties



wordSelector.properties

words.txt

E' il file di parole fornito. Contiene 30824 parole, tutte lunghe 10 caratteri, che verranno periodicamente scelte dal WordSelector come nuova parola del giorno. E' il file indicato di default dentro "wordSelector.properties", ma si può scegliere un file diverso, affinché

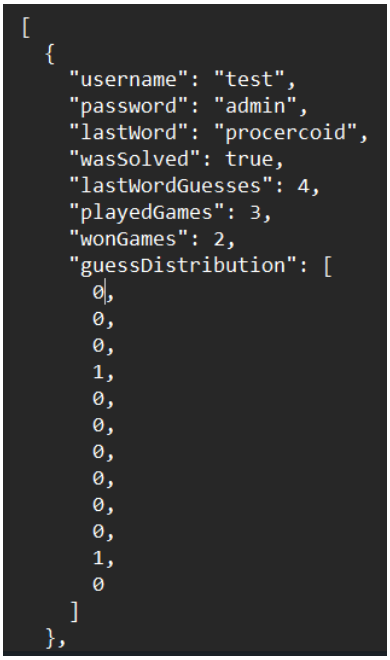
contenga parole lunghe 10 caratteri e ordinate in ordine alfabetico (necessario per la ricerca binaria delle parole, di cui discuteremo dopo).

accounts.json

E' il file che usiamo per salvare i dati degli utenti in formato JSON.

A destra, la struttura dell'oggetto JSON di un qualsiasi utente.

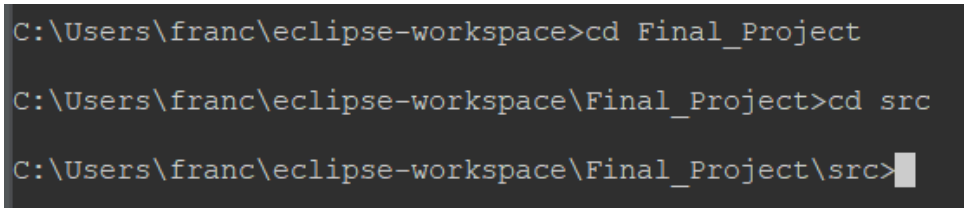
ATTENZIONE: Come discuteremo dopo, al primo avvio del server è necessario che questo file contenga due parentesi quadre [ ]
per permettere al JsonReader di "consumarle" come token e non avere un errore.



Come compilare e avviare Wordle 3.0

Dopo aver visto la struttura del progetto, per compilare e avviare l'applicazione è necessario seguire i seguenti passi:

1. Accedere alla cartella del progetto e in particolar modo entrare nella cartella src del progetto, che contiene i file .java. (Si può usare il comando cd per spostarsi tra le cartelle).
   Il path dovrebbe avere un formato simile a questo:

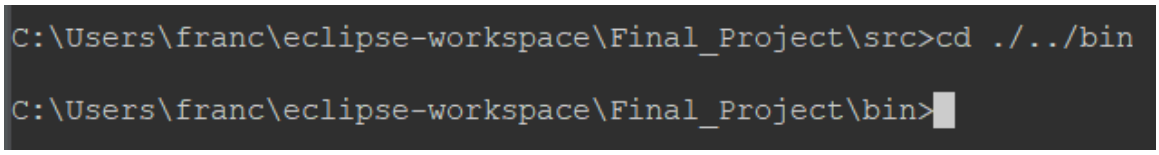


2. Su un terminale (es. “Windows Powershell” o “CMD” su Windows, o direttamente dal terminale di Eclipse, VSCode, …) digitare il seguente comando:

```
javac -cp ".;gson-2.8.6.jar" *.java -d ./../bin
```

   Il significato di questo comando è che la compilazione deve includere nel class path il file “gson-2.8.6.jar” presente in questa cartella (.;), bisogna compilare tutti i file “.java” e l’esito dell’operazione ha destinazione nella cartella “./../bin”

3. Dopo aver compilato i file .java, otteniamo i file eseguibili .class . La loro destinazione però non è src, quindi per avviarli bisogna entrare dentro la cartella bin. Possiamo fare ciò mediante il comando “cd ./../bin”
   Di seguito un esempio:



Si noti il cambiamento della cartella: non siamo più in src, bensì in bin.

4. **ATTENZIONE:** E’ bene verificare, prima di avviare i file, che nel file “accounts.json” siano presenti due parentesi quadre. Il funzionamento del programma non è garantito se dovessero mancare le parentesi, in quanto l’utilizzo di un JsonReader per la lettura del file JSON ne richiede la presenza.

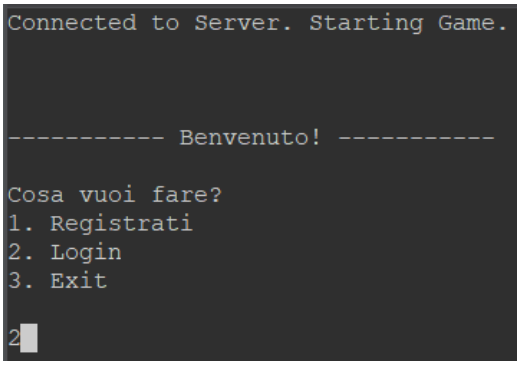
5. Dopo essersi posizionati in bin, possiamo far partire i .class precedentemente creati. Questa operazione si può eseguire mediante i seguenti comandi:

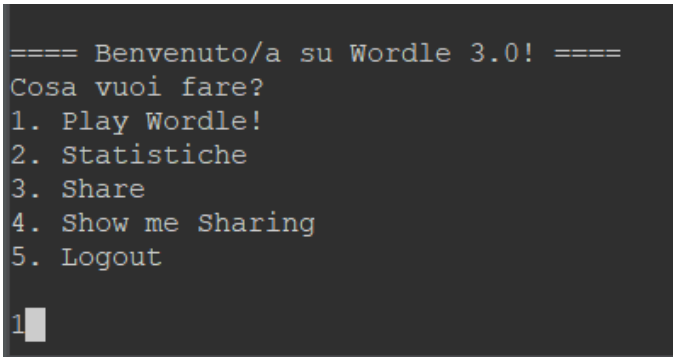
```
java -cp ".;gson-2.8.6.jar" ServerMain

java -cp ".;gson-2.8.6.jar" ClientMain
```

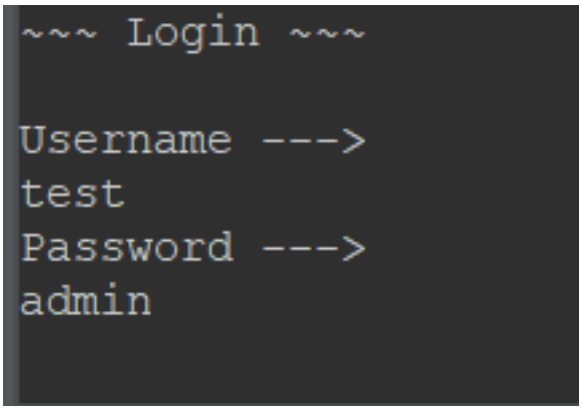
   Il significato di questi comandi è che si deve eseguire i file ServerMain e/o ClientMain includendo nel ClassPath lo stesso file gson.jar visto in precedenza, e che è presente anche nella cartella bin. E’ bene, per quanto non impossibile, non avviare gli altri eseguibili, in quanto strettamente collegati a ServerMain e ClientMain, e insensati se eseguiti da soli.

5. Una volta avviati i due file, si potrà interagire facilmente e intuitivamente col Client, che manderà le richieste adeguate al Server.
   Per interagire col Client sarà necessario dare in input il numero E SOLO IL NUMERO corrispondente all'azione che si vuole effettuare. Un input errato non verrà considerato e il Client richiederà di nuovo l'input all'utente.





Esempio di utilizzo del Client



In alcune fasi è possibile (e necessario) scrivere parole e caratteri piuttosto che utilizzare i numeri per navigare i vari menù. Come si vede nell'esempio, nelle fasi di Login, Registrazione o di Gioco, nel tentativo di indovinare la parola corretta, un input di tipo testuale sarà accettato ed elaborato di conseguenza.

Strutture Dati utilizzate e Scelte Implementative

Quando si lavora in un paradigma Client-Server in Java, esistono due strategie principali, che fanno uso di strutture dati diverse. La prima strategia è utilizzare Java I/O, quindi utilizzare gli Stream classici per la comunicazione mediante le Socket. La seconda è quella di utilizzare Java NIO, quindi usare i Channel e i Selector per la

gestione delle richieste dei Client. Per familiarità personale, la prima soluzione è risultata la più efficace.

Memorizzazione degli utenti

Gli utenti vengono memorizzati in un file JSON. L'accesso al file avviene mediante un JsonReader della libreria GSON. E' necessario che il file contenga delle parentesi quadre [ ], in quando il JsonReader deve consumarle come token per poter iniziare a parsare il file JSON. E' quindi necessario che al primissimo avvio del server, il file contenga già le parentesi, nonostante il server non sia mai stato avviato, per non avere errori nel parsing (Questo è dovuto ai metodi .beginArray() ed .endArray())

```
json_in.beginArray();
while (json_in.hasNext()) {
    Account account = gson.fromJson(json_in, Account.class);
    users.add(account);
}
json_in.endArray();
```

Si noti l'utilizzo dei metodi beginArray e endArray, che "consumano" le parentesi quadre.

Registrazione, Login e Logout

La registrazione, il login e il logout fanno uso del solito JsonReader per accedere al file, caricare gli utenti in una lista e utilizzarla per eseguire le operazioni che occorre eseguire.
Nel progetto, queste fasi non sono ottimizzate: per ogni operazione di login, registrazione o logout, viene caricata una lista che al termine delle operazioni viene "scaricata". Una soluzione più efficiente sarebbe quella di caricare all'avvio del server una lista di tutti i giocatori, o meglio ancora, di accedere al file dei giocatori volta per volta, senza doverlo caricare in memoria. Tuttavia, poiché ci troviamo in un ambiente di sviluppo abbastanza piccolo, la soluzione adottata risulta essere abbastanza efficiente da non causare colli di bottiglia.

Estrazione della Secret Word

L'estrazione della parola segreta avviene mediante un RandomAccessFile. Questo risulta molto utile in quando la scelta della parola deve essere casuale e l'utilizzo di un

RandomAccessFile ci permette di accedere randomicamente al file mediante la funzione “seek()”.

Progressi nella fase di gioco

Durante la fase di gioco, cioè quando l’utente sta cercando di indovinare la parola segreta, manteniamo una HashMap per contare le occorrenze (mancanti) di ogni lettera della parola.

Lettura delle parole dal file

Si sfrutta l’ordinamento delle parole all’interno del file per accedere e ricercare una parola (per determinare la sua esistenza all’interno della lista) mediante una ricerca binaria.

Gestione dei Thread

Si usa un ThreadPool di tipo Cached, in modo che vi sia un’associazione uno ad uno tra Client che arrivano e Thread che li “servono”. Non diamo alcun limite alla dimensione del pool.

Salvataggio Notifiche

Utilizziamo un ArrayList per memorizzare le notifiche dentro il thread “MulticastHandler”. Usiamo i metodi predefiniti sugli ArrayList per aggiungere le notifiche alla coda.
Il fatto che un Client possa condividere più volte l’esito di una partita e questo venga condiviso più volte nella rete Multicast è un comportamento voluto. (Si basa sul pensiero che anche sulle social network più conosciute come Facebook, un utente può condividere più volte l’esito di una partita, e comparirà più volte sulla bacheca di coloro che hanno un legame con quell’utente).

Generazione dei Thread

A lato **Client**, dopo aver effettuato il login viene lanciato un thread (multicastHandler) che rimane in ascolto di eventuali notifiche provenienti dal gruppo di multicast. E’ importante che venga avviato dopo il login, in quanto le notifiche sono “personalizzate” per ogni utente e non si trovano su una struttura dati condivisa a tutti gli utenti.
Per ciò che riguarda il **Server**, la struttura è leggermente più complessa, ma comunque semplice. All’avvio, la classe “ServerMain” crea un CachedThreadPool, in modo da

poter generare un thread per ogni Client che si connette. Prima ancora di accettare i Client però, crea un ulteriore thread a parte, che sarà il responsabile della scelta delle parole (WordSelector). Sia il Server che il WordSelector sono da intendersi sempre attivi. Il WordSelector al suo interno utilizza due classi della libreria “java.util”: Timer e TimerTask. Similarmente al funzionamento del CachedThreadPool e dei task di “Server_task”, la classe TimerTask definisce il task schedulato, nel nostro caso la scelta della parola, mentre la clase Timer gestisce i thread (uno solo nel nostro caso) sui quali sono eseguiti i task.

Creazione e avvio dei file “.jar”

Per la creazione dei file Jar, si è proceduto con l’utilizzo di un terminale a riga di comando, piuttosto che con la funzione “Export” di Eclipse. La procedura adottata è la seguente:

1. Una volta aperto un terminale (es. PowerShell, CMD, ecc.), ci si sposta nella cartella contenente i file “.class” che vogliamo includere nel file Jar. Nel nostro caso, ci spostiamo nella cartella “bin” del progetto.
   Il path dovrebbe essere qualcosa di simile a questo:

```
PS C:\Users\franc> cd .\eclipse-workspace\Final_Project\bin\
PS C:\Users\franc\eclipse-workspace\Final_Project\bin> ls


    Directory: C:\Users\franc\eclipse-workspace\Final_Project\bin


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         27/04/2023     09:55           1024 Account.class
-a----         28/04/2023     11:07           1918 accounts.json
-a----         27/04/2023     09:53           1193 Checker.class
```

2. Una volta all’interno della cartella, si fornisce questo comando `jar cf jar-file input-file(s)`, dove “jar-file” è il nome del file che si sta creando, e gli “input-files” sono i file che vogliamo includere nel Jar. Nel nostro caso, il comando ha una struttura simile alla seguente:

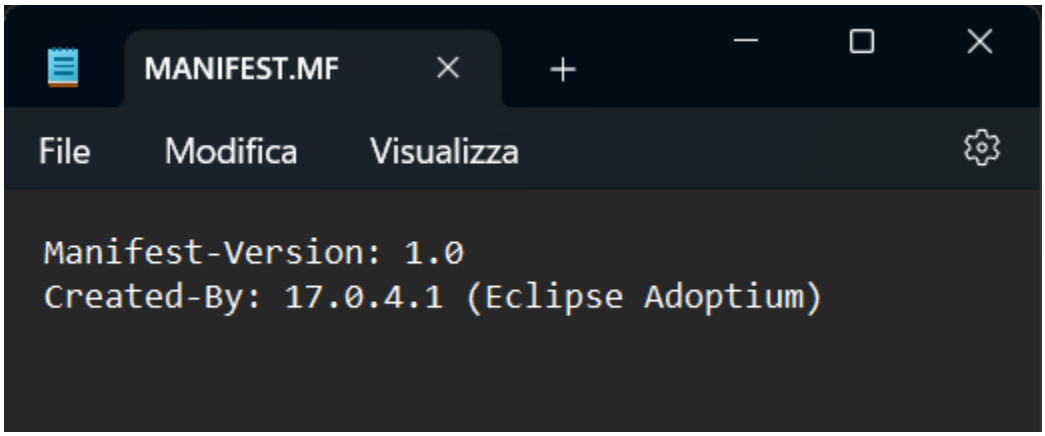
```
PS C:\Users\franc\eclipse-workspace\Final_Project\bin> jar cf ClientTest.jar ClientMain.class
 clientAndServer.properties MulticastHandler.class
```

Creazione del Jar per il Client

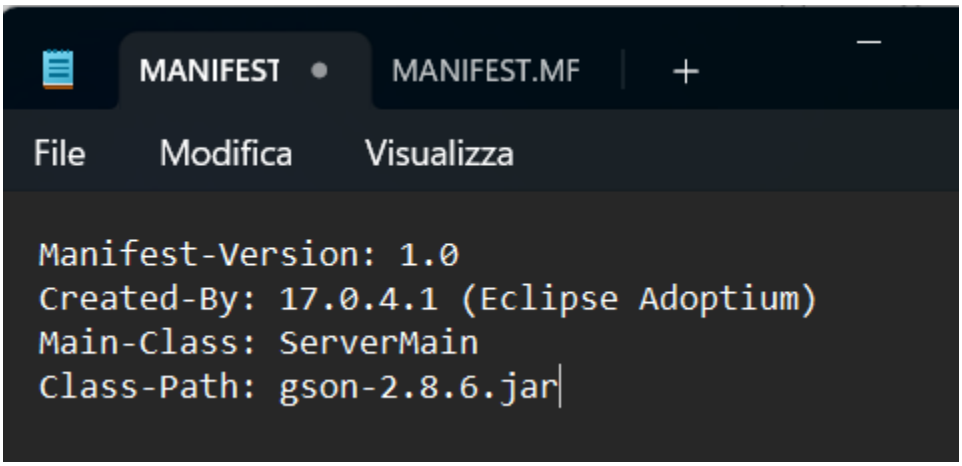
```
PS C:\Users\franc\eclipse-workspace\Final_Project\bin> jar cf ServerTest.jar ServerMain.class
 clientAndServer.properties Account.class accounts.json Checker.class Server_Task.class words
.txt WordSelector.class wordSelector.properties 'WordSelector$1.class'
PS C:\Users\franc\eclipse-workspace\Final_Project\bin>
```

Creazione del Jar per il Server

3. Una volta creati i Jar, andiamo a modificare il file Manifest presente al loro interno. Ci basta aprire il file .jar con un gestore di archivi come WinRar.
   a. Entrati nel file .jar, accediamo alla cartella “META-INF” all’interno del quale sarà presente il file “MANIFEST.MF”
   b. Apriamo il file con un editor di testo (es. Blocco Note). Di fronte a noi avremo un file di questo tipo:



   c. Modifichiamo il file aggiungendo la Main-Class (cioè la classe che contiene il metodo main da avviare all’avvio del jar) e il ClassPath (che contiene le librerie esterne utilizzate). Nel nostro caso il file dovrà contenere le seguenti informazioni:



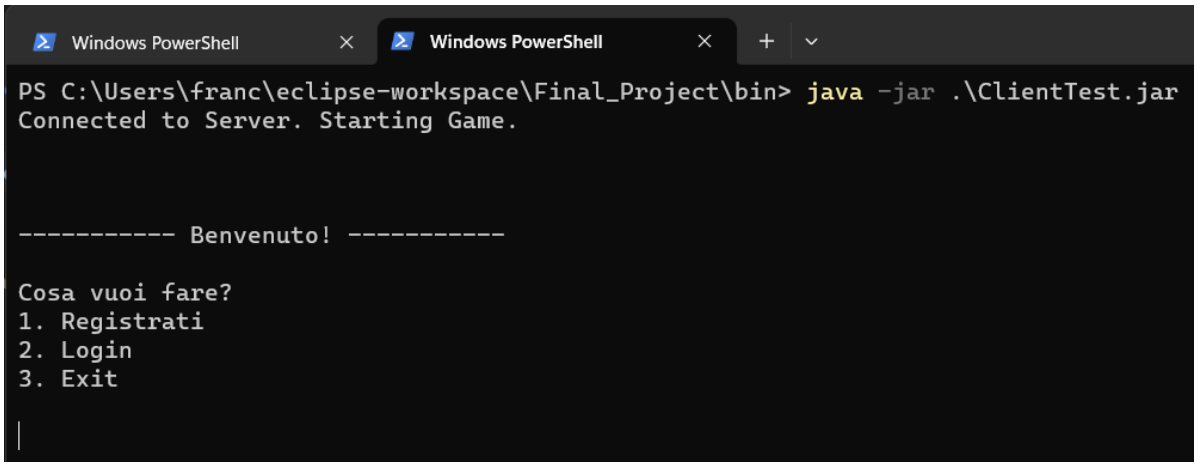
Esempio del Manifest lato Server

d. Ripetiamo la procedura anche lato Client.

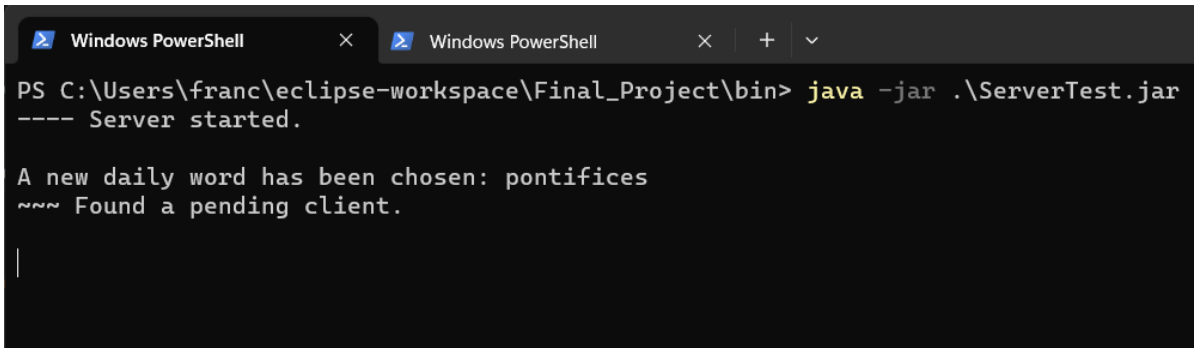
4. Una volta creati i Jar, questi si troveranno nella cartella bin, assieme alle altre classi. Non resta altro che avviarli e testare il funzionamento mediante il comando

```
java -jar .\ClientTest.jar
java -jar .\ServerTest.jar
```

Di seguito, i programmi all'avvio:



Avvio del Client



Avvio del Server

N.B: Questa procedura è già stata svolta. I file Jar si trovano già nella cartella “bin” del progetto. Per avviarli, non rimane che entrare nella cartella ed eseguirli coi comandi mostrati sopra.

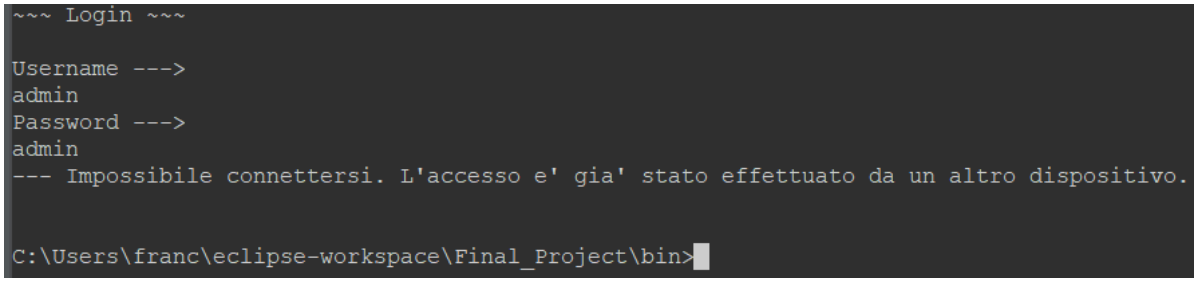
Accorgimenti Vari

1. Si noti che qualsiasi parola, numero o carattere scritto sulla riga di comando è case sensitive. Inoltre, qualsiasi spazio posto all’inizio o alla fine dell’input non determina un comportamento diverso da quello dove l’input risulta senza gli spazi. Ove possibile è stato utilizzato il metodo “.trim()” che elimina eventuali whitespace prima e dopo la parola intesa, e il metodo “contains()”, adottato per controllare la presenza di spazi all’interno delle comunicazioni.

```
81                  //Richiesta Username
82                  System.out.println("Username --> ");
83                  String username = keyboard_in.nextLine().trim();
84                  if(username.contains(" ")) {
85                      System.out.println("Lo username non può contenere spazi.\n");
86                      continue;
87                  }
88                  //Richiesta Password
89                  System.out.println("Password --> ");
90                  String password = keyboard_in.nextLine().trim();
91                  if(password.contains(" ")) {
92                      System.out.println("La password non può contenere spazi.\n");
93                      continue;
94                  }
95
```

Alle righe 83 e 90 viene invocato il metodo “trim()”. Alle righe 84 e 91 viene controllata la presenza di whitespace all’interno della stringa mediante il metodo “contains()”.

2. Il sistema, mediante la classe “Checker” riesce ad identificare se un utente è online, in quanto mantiene una lista degli utenti attivi. Se un utente prova a fare l’accesso da un secondo terminale mentre è già loggato, verrà notificato del fatto che l’account è già online e l’esecuzione del programma verrà terminata.



Esempio di esecuzione di “Client” quando un’altra istanza è loggata con lo stesso account.