

Reti di calcolatori e laboratorio

WordQuizzle

Marco Antonio Corallo, 531466

1. Introduzione
2. Architettura e moduli
 1. Architettura
 2. Server
 3. Client
 4. Servizio di registrazione
3. Threads e concorrenza
4. Guida per l'uso
 1. Librerie esterne
 2. Compilazione ed esecuzione
 3. Comandi e interfaccia
5. Test e risultati

Introduzione

Il modulo di Laboratorio del corso di *Reti di calcolatori* prevede lo svolgimento di un progetto da sviluppare utilizzando il linguaggio Java in ambiente Linux.

Il progetto deve essere svolto singolarmente da un solo studente.

Questo documento descrive la struttura complessiva del progetto *WordQuizzle*.

La scadenza per la consegna del progetto coincide con la scadenza dell'appello straordinario (ottobre/novembre 2020) dell'a.a 2019/2020.

Il progetto dovrà essere sviluppato e documentato utilizzando gli strumenti, le tecniche e le convenzioni presentate durante il corso. L'obiettivo del progetto è la realizzazione di un sistema *client-server*, che permette l'interazione fra coppie di clients al fine di sfidarsi in gare di traduzioni italiano-inglese del maggior numero di parole fornite dal server.

Il sistema consente inoltre la gestione di una rete sociale tra gli utenti iscritti.

Architettura e moduli

Architettura

La specifica prevede che il progetto sia realizzato come sistema client-server. La comunicazione di operazioni, interrogazioni, esiti e risposte avviene comunemente tramite il protocollo **TCP**, con un paio di eccezioni.

La registrazione al servizio, infatti, avviene tramite **chiamata di procedura remota (RPC)**, supportata da Java tramite **RMI: Remote Method Invocation**.

Viene invece utilizzato il protocollo **UDP** nel momento in cui il server intende inoltrare una richiesta di sfida ad un giocatore, che chiameremo giocatore *sfidato*, o giocatore *passivo*.

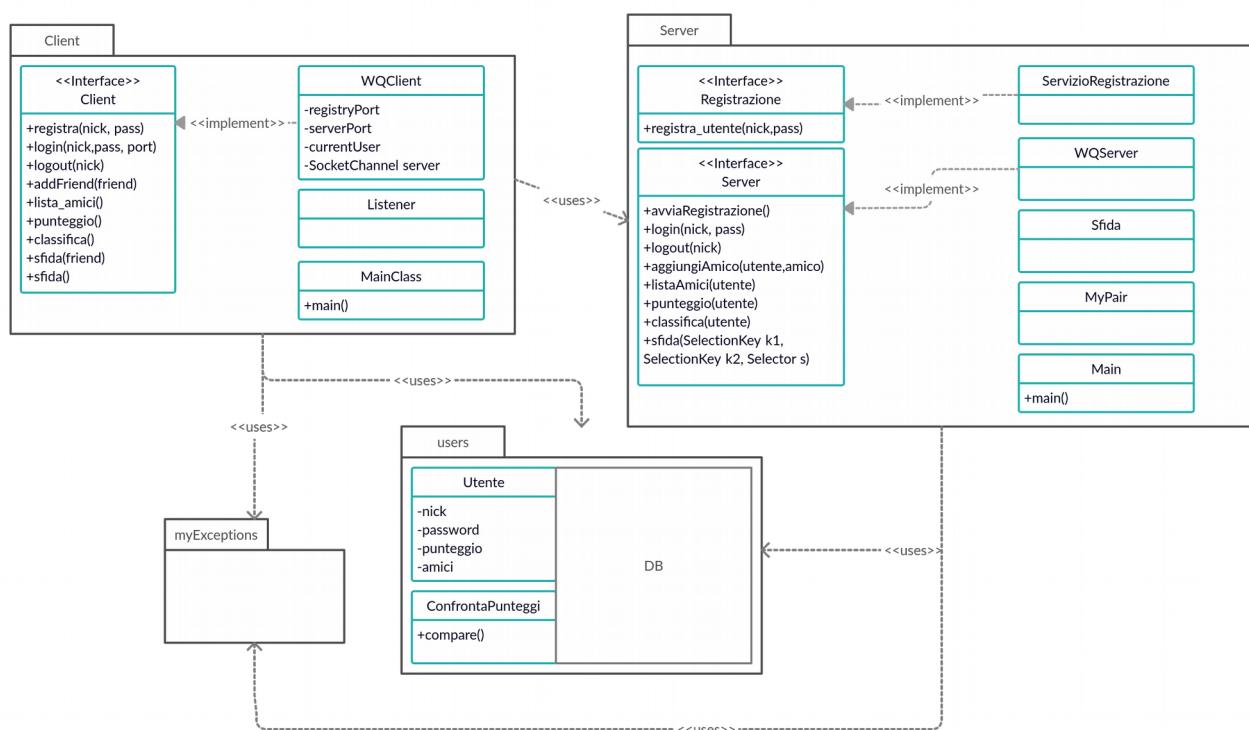
Per sua natura, infatti, il protocollo UDP non è particolarmente adatto per altri utilizzi richiesti nel progetto.

Durante l'implementazione del sistema sono state seguite le linee guida della progettazione **Secure by Design**, applicando i concetti della **Defensive Programming** sin dall'interfaccia dei moduli principali.

Ciò non è sinonimo di un sistema sicuro, privo di bugs o difetti, ma identifica gli errori di progettazione più comuni in modo da poterli gestire.

I controlli sulla correttezza dei parametri, ove possibile, vengono fatti dal Client; vengono però sempre integrati con controlli server-side,volti ad evitare possibili malfunzionamenti e crash del server.

Il sistema è stato diviso in *packages*, qui riassumibile in **UML** ed accuratamente approfondito in seguito.



Server

All'interno di questo package si trova l'interfaccia omonima, che definisce i metodi previsti dalla specifica del progetto e che trova implementazione nella classe *WQServer*.

Tutti i metodi dell'interfaccia richiedono parametri tipici per l'utilizzo di un oggetto di tipo Server, ad eccezione di **sfida**, che richiede come parametro le *SelectionKey* dei due giocatori ed il *selettore* su cui sono registrate.

Questo evidenzia un apparente *disaccoppiamento* non totale dall'implementazione, facilmente superabile però tramite il passaggio di oggetti effimeri.

Infatti, l'implementazione del metodo fa uso della classe *Sfida*, spiegata poco sotto, che prevede questi parametri.

All'interno del package sono presenti anche l'interfaccia e relativa implementazione del servizio di registrazione, facente uso di *RMI*, che viene trattato successivamente, la classe *Sfida* e la classe *MyPair*.

La prima delle due implementa *Runnable*, rappresenta infatti un task che viene sottomesso ad un *ThreadPool* memorizzato all'interno della classe *WQServer* e che si occupa, in un flusso di esecuzione a parte, della sfida di due utenti.

Il costruttore inizializza le informazioni sui due utenti e si occupa di reperire e tradurre 10 parole da un dizionario di 150.

Successivamente, alloca lo spazio necessario per inviare e ricevere le parole e registra le *SelectionKey* passate come parametro in un selettore **locale**.

Il metodo prevede infatti che le chiavi vengano cancellate dal selettore passato come parametro poco prima dell'invocazione, per poi registrarle nuovamente prima di terminare l'esecuzione del thread.

Il selettore locale fa uso di *timeout*, per segnalare che i 30 secondi concessi per la sfida sono terminati.

Durante la sfida le parole proposte, le risposte e i punteggi vengono salvati in un file temporaneo denominato *Utente1ANDUtente2.txt* in modo da poter consultare i *log* qualora dovesse essere utile o necessario farlo.

Una volta identificato il vincitore viene inviato l'esito della sfida, con relativo punteggio, ai due giocatori e viene modificato il *database* con i punteggi aggiornati.

La classe *MyPair* è invece una semplice struttura dati che permette di appaiare *<nomeUtente, numeroPorta>* al momento del login.

Questa scelta è dovuta alla necessità del server di conoscere il numero di porta su cui un client è in ascolto di **datagrammi**, che indicano la ricezione di una sfida.

La classe emula il comportamento della classe **javafx.util.Pair**, in modo tale da evitare l'importazione della libreria esterna FX a *compile-time*.

Client

All'interno del package si trovano, come per l'antitetico appena esposto, l'interfaccia e l'implementazione: rispettivamente *Client* e *WQClient*.

Questi moduli presentano, tra i normali metodi richiesti dalla specifica e che comunicano al server l'intenzione di richiedere una determinata operazione, l'*overload* del metodo *Sfida*.

Questo, che a differenza dell'omonimo metodo non richiede parametri, viene richiamato direttamente dal client del giocatore sfidato e che ha accettato di giocare e, indirettamente, dal client del giocatore sfidante. Quest'ultimo infatti lo richiama non appena riceve risposta positiva dal giocatore passivo.

Vi è poi una classe *Listener*, implementazione di un *Runnable* che viene istanziata ed eseguita immediatamente prima di effettuare il login.

Il thread listener genera un numero pseudocasuale nell'intervallo [50000,65000] e controlla se la rispettiva porta è libera. Il client passa quindi questo numero al server, insieme alle informazioni utente per il login, dopodiché il thread si mette in ascolto di datagrammi UDP.

Allorquando questo si verifica, l'oggetto listener – che conserva un puntatore al main thread – scatena un'*interruzione* sul thread principale.

Quest'ultimo riesce a catturare l'interruzione utilizzando, durante l'interazione utente, un oggetto di tipo *BufferedReader* che possiede un metodo *ready()* per verificare che la sorgente di input sia pronta.

Finché non lo è, controlla che non siano arrivate interruzioni.

Servizio di registrazione

Interfaccia ed implementazione del metodo di registrazione risiedono all'interno del package *server* ma, come accennato prima, utilizzano RMI piuttosto che socket TCP **esplicativi** durante la comunicazione.

Il metodo *WQServer.avviaRegistrazione()* si occupa di allocare l'oggetto remoto, esportarlo, creare il *registry* e pubblicarvi lo *stub*.

Nel momento in cui un utente intende registrarsi al servizio, il client mette invece a disposizione il metodo *WQClient.registra(utente, password)*, che individua lo stub e lo utilizza per richiamare il metodo remoto di registrazione.

Threads e concorrenza

La scelta progettuale fatta in merito alla concorrenza ed al multithreading è di sfruttare, oltre che il main thread ed i vari threads RMI gestiti dalla JVM, un **thread per ogni sfida**.

I motivi alla base di questa scelta si possono trovare innanzitutto nel guadagno di *semplicità di progettazione e manutenzione*, dato che la sfida rappresenta di per sé una parte rilevante del sistema in termini di responsabilità e complessità.

Inoltre, come visto durante il corso, vi è una motivazione di progettazione non trascurabile: un **singolo thread** per tutti i clients ha *scalabilità nulla*, *accept latency* alta e *reply latency* bassa; al contempo lanciare un **thread per ogni connessione** può portare a bassa *reply latency* e bassa *efficienza*.

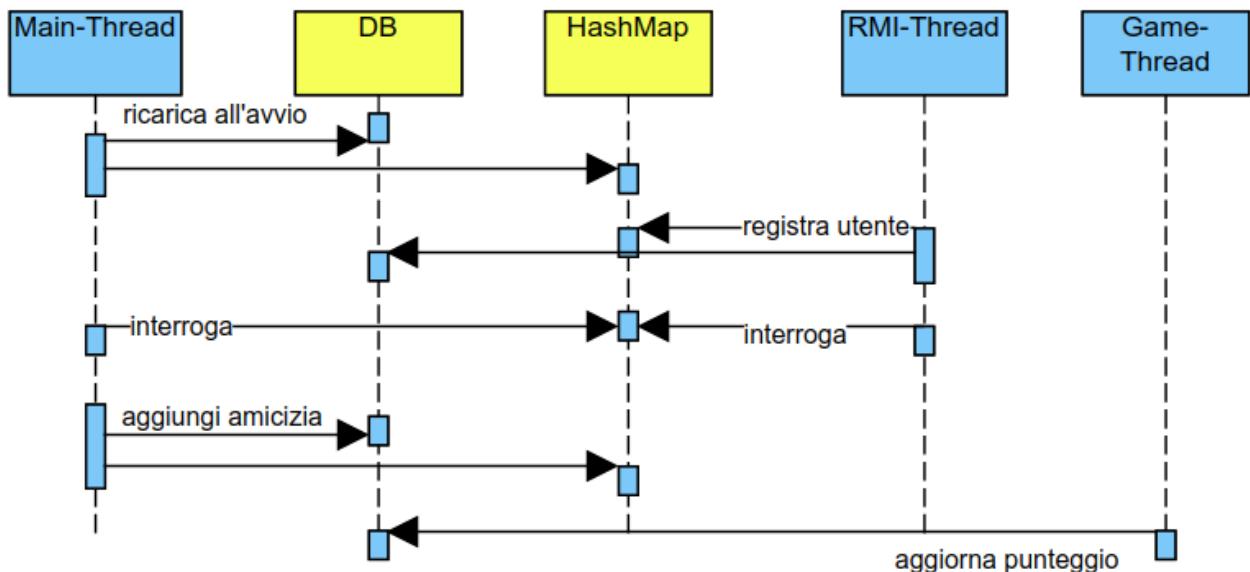
La decisione presa rappresenta quindi un punto d'incontro tra le due soluzioni: le connessioni vengono gestite da un unico thread per operazioni semplici, quando cioè il *tempo di servizio* di un singolo utente è garantito rimanere basso, mentre si suddivide il carico di lavoro su più flussi di esecuzione se l'operazione da eseguire richiede tempo e complessità non trascurabile.

Vi è quindi un miglioramento di performance e scalabilità

Per la gestione di questi threads viene utilizzato un *cached thread pool*: questa scelta è dettata principalmente dall'*usabilità* del sistema: due utenti che avviano una sfida non possono né vogliono aspettare che un'altra coppia di utenti finisca di giocare. È preferibile scalare maggiormente il sistema piuttosto che aumentare l'*acceptance latency*.

Per quanto concerne le **risorse condivise**, il problema è limitato alle strutture per la memorizzazione degli utenti: una *mappa* memorizzata all'interno del server, cui accedono anche i thread RMI, ed il *database JSON*, condiviso anch'esso con i thread RMI e con i thread di sfida.

Una semplificazione delle possibili interazioni con le risorse, da parte dei threads, è illustrata in seguito tramite un *sequence diagram*.



Per evitare problemi di concorrenza su queste risorse, sono state applicate contromisure semplici ma efficaci:

- Piuttosto che un *HashMap* è stato utilizzato un oggetto della classe *ConcurrentHashMap*, consigliato dalla documentazione rispetto ad *HashTable* e su cui è garantita la sincronizzazione;
- È stato utilizzato (acquisita/rilasciata) un oggetto *Lock* per ogni accesso al database. Questa scelta, sicuramente non la più efficiente considerato il noto caso lettori/scrittori, è motivata solamente dalla semplicità di progettazione. Le operazioni su DB, infatti, con l'eccezione del main thread, vengono eseguite esattamente una volta per ogni thread.

Guida per l'uso

Librerie esterne

Per la gestione del database JSON è stata utilizzata una libreria esterna, nota come **JSON.simple**, già illustrata durante il corso.

La libreria è presente all'interno dell'archivio, nella directory principale ed è necessario aggiungerla al *classpath* per poter compilare il progetto correttamente.

Compilazione ed esecuzione

Per semplificare il processo di *building* del progetto è stato scritto un brevissimo *makefile* tramite cui compilare ed eseguire uno o più moduli.

Istruzioni:

- **make** : compila ordinatamente i moduli, creando i file .class;
- **make server** : esegue un'istanza del server;
- **make client** : esegue un'istanza del client;
- **make clean** : elimina i file .class ed eventuali file di log delle sfide;

Comandi e interfaccia

Il client offre una semplice interfaccia a linea di comando, quasi uguale a quella proposta nel testo di specifica del progetto. Di seguito l'*usage*, consultabile anche passando il parametro *--help* al programma client.

```
usage : COMMAND [ ARGS ... ]  
Commands:  
registra <nickUtente> <password>: registra l'utente (MAX 10  
caratteri)  
login <nickUtente> <password> : effettua il login  
logout : effettua il logout  
aggiungi <nickAmico> : crea relazione di amicizia con nickAmico  
amici : mostra la lista dei propri amici  
sfida <nickAmico> : richiesta di una sfida a nickAmico  
punteggio : mostra il punteggio dell'utente  
classifica : mostra una classifica degli amici dell'utente  
(incluso l'utente stesso)
```

Test e risultati

Il sistema è stato testato, nei giorni precedenti alla consegna, su molti casi limite tramite *valori di frontiera*, nonché su casi comuni di interazione utente.

È stato proposto per alcuni minuti a diversi utenti, sperimentando tipiche interazioni di gioco ed interrogazione, oltre che verificato, tramite *black-box testing*, da pochi colleghi e dall'autore.

Sono stati testati anche i casi di *timeout*, di *sfide parallele* e di richieste inviate durante una partita già iniziata.

I test hanno aiutato a rilevare alcuni bugs, ma...

“Program testing can be used to show the presence of bugs, but never to show their absence!”

E. Dijkstra



```
Attività □ Terminale □ dom 25 ott. 9.07 marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze
File Modifica Visualizza Cerca Terminale Aiuto
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: libertà
Richesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: ricordo
Purtroppo hai perso. Punteggio: -3
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: fatto
Purtroppo hai perso. Punteggio: -2
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: modo
Purtroppo hai perso. Punteggio: -12
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: politica
Purtroppo hai perso. Punteggio: -14
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: titolo
Purtroppo hai perso. Punteggio: -16
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: sviluppo
Purtroppo hai perso. Punteggio: -18
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: destino
Purtroppo hai perso. Punteggio: -20
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: stato
Purtroppo hai perso. Punteggio: -22
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: fatto
Colpo di scena: pareggio! Siete entrambi vincitori, con il punteggio di 0, ma senza bonus!
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
classifica
classifica:
[{"Parola": "modo"}, {"Parola": "titolo"}, {"Parola": "sviluppo"}, {"Parola": "destino"}, {"Parola": "stato"}]
["*Paperoga": -2], ["*Gastone": -7]
Cosa vuoi fare? [help] per vedere la lista comandi.
logout
Chiusura in corso...
Marco e Gastone sono ora anici.
Marco e Gastone sono ora anici.
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Marco e Gastone sono ora anici.
Marco e Gastone sono ora anici.
Sfida lanciata! In attesa della risposta da Gastone
Il tuo amico ha rifiutato la sfida.
Cosa vuoi fare? [help] per vedere la lista comandi.
logout
Chiusura in corso...
Marco si è stappato correttamente.
Marco si è stappato correttamente.
Marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze$ marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze
File Modifica Visualizza Cerca Terminale Aiuto
Cosa vuoi fare? [help] per vedere la lista comandi.
anici
Lista amici:
Paperoga
Paperino
*****...
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
n
Cosa vuoi fare? [help] per vedere la lista comandi.
logout
Chiusura in corso...
Gastone si è stappato correttamente.
Marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze$ marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze
File Modifica Visualizza Cerca Terminale Aiuto
Avete 30 secondi, per rispondere a 10 domande!
Parola: fatto
Colpo di scena: pareggio! Siete entrambi vincitori, con il punteggio di 0, ma senza bonus!
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
classifica
classifica:
[{"Parola": "modo"}, {"Parola": "titolo"}, {"Parola": "sviluppo"}, {"Parola": "destino"}, {"Parola": "stato"}]
["*Paperoga": -2], ["*Gastone": -7]
Cosa vuoi fare? [help] per vedere la lista comandi.
logout
Chiusura in corso...
Paperoga si è stappato correttamente.
Marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze$ marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze
File Modifica Visualizza Cerca Terminale Aiuto
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
Accettare? [s/n]
s
La sfida sta per iniziare...
Avete 30 secondi, per rispondere a 10 domande!
Parola: modo
Colpo di scena: pareggio! Siete entrambi vincitori, con il punteggio di 0, ma senza bonus!
Cosa vuoi fare? [help] per vedere la lista comandi.
Richiesta arrivata.
punteggio
punteggio: -41
Cosa vuoi fare? [help] per vedere la lista comandi.
logout
Chiusura in corso...
Paperino si è stappato correttamente.
Marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze$ marco@marco-Lenovo-ideapad-300-15ISK:~/Scrivania/WordQuizze
```