Relazione progetto di laboratorio di reti

WORTH: WORkTogetHer

A.A. 2020/2021

Rachele Puri 547938

Introduzione

WORTH nasce con l'idea di implementare un metodo di gestione "agile", il metodo Kanban. Questo strumento di gestione mira ad aiutare i team nell'organizzazzione per la realizzazione di uno o più progetti. In questo specifico caso WORTH sarà il server al quale i vari client potranno collegarsi per interagire con esso. I vari utenti, attraverso linea di comando, digiteranno dei comandi che il client si occuperà di far arrivare al server per soddisfare la specifica richiesta. Il server dopo aver ricevuto la richiesta, invierà una risposta che l'utente potrà visualizzare tramite terminale. Di seguito sono riportate in maniera più approfondita le varie interazioni sopra citate e le varie strutture dati utilizzate.

Compilazione ed esecuzione

Una volta all'interno della cartella "worth" sarà possibile compilare il server e il client attraverso due comandi:

- *javac ./utility/*.java ./server/*.java*
- *javac ./utility/*.java ./client/*.java*

Per l'esecuzione invece sono necessari altri due due comandi:

- *java server.ServerMain*
- *java client.ClientMain*

Implementazione

All'interno della cartella *worth* sono presenti tre cartelle principali:

- utility
- server
- client

Utility

In questa cartella sono presenti tutte quella classi, e tutte quelle interfacce, utili per creare le strutture dati sulle quali il server salverà i vari dati e attraverso le quali potrà soddisfare le richieste da parte del client. Le classi e le interfacce presenti sono le seguenti:

- *Chat.java* : definisce la "chat" di un progetto specifico in cui i vari utenti potranno scambiarsi messaggi, più precisamente datagrammi. Un oggetto di tipo Chat comprende il nome del progetto a cui si riferisce, l'indirizzo IP multicast e la porta UDP (IP+porta permettono di creare il canale per lo scambio di datagrammi).
- *Chats.java* : definisce la lista di tutte le chat presenti all'interno del sistema. Questa lista è rappresentata come una *ConcurrentHashMap*, dove la chiave corrisponde al progetto e il valore corrisponde alla chat relativa a quel progetto;
- *Database.java* : definisce tutte le varie strutture dati utilizzate dal server. In particolare al suo interno troveremo tre *ConcurrentHashMap*: la prima utilizzata per memorizzare tutti i progetti creati, la seconda per memorizzare l'associazione progetto-membri, e la terza per memorizzare l'associazione progetto-card.
- *Utente.java* : definisce appunto l'utente attraverso uno username univoco, una password con i quali potrà registrarsi al servizio e successivamente fare il login o il logout, uno stato (online o offline) e una lista di progetti a lui associati.
- *Userdatabase.java* : definisce la lista di tutti gli utenti presenti nel sistema. Questa lista è rappresentata con una *ConcurrentHashMap*, dove la chiave corrisponde allo username e il valore corrisponde all'oggetto di tipo Utente.
- *IPGenerator.java :* genera un indirizzo IP utilizzato per la connessione UDP per lo scambio dei messaggi nelle varie chat.
- *UDPportGenerator.java :* genera la porta UDP utilizzata, insieme all'indirizzo IP, per lo scambio dei messaggi nelle varie chat.
- *ClientNotificeInterface.java* : interfaccia utilizzata per realizzare, lato client, il servizio di notifica per gli aggiornamenti sullo stato degli utenti registrati. Il servizio è implementato con il meccanismo RMI.
- *ServerNotificeInterface.java* : interfaccia utilizzata per realizzare, lato server, il servizio di notifica per gli aggiornamenti sullo stato degli utenti registrati. Il servizio è implementato con il meccanismo RMI.
- *RegistrazioneInterface.java* : interfaccia utilizzata per effettuare, lato server, la registrazione di un utente al servizio.
- *UpdateHashmapInterface.java* : interfaccia utilizzata per realizzare, lato server, il servizio di notifica per gli aggiornamenti sulle varie strutture dati da comunicare al client.

- *UpdateClientInterface.java* : interfaccia utilizzata per realizzare, lato client, il servizio di notica per gli aggiornamenti sulle varie strutture in modo da poter aggiornare le proprie strutture.

Server

Questa cartella comprende la classi principali del server che permettono l'interazione con il client. Le classi presenti sono le seguenti:

- *ServerMain.java* : all'interno di essa avviene la realizzazione del servizio di notificata attravero la creazione dell'RMI, la creazione delle varie strutture dati sopra citate, il ripristino dei dati e l'avvio del server. Sono presenti all'interno anche tutti quei metodi utili per rendere il sistema persistente.
- *ServerTCP.java* : corrisponde al server vero e proprio, mandato in esecuzione nella classe *ServerMain.java* attraverso il metodo *.start()*. All'interno di questa classe viene avviata la connessione TCP attraverso la quale potrà comunicare con i vari client. Dopo aver stabilito la connessione, avviene la lettura e la scrittura sulla socket. Il server legge sulla socket i vari comandi e manda sulla socket le rispettive risposto. Qui viene gestito il caso in cui il comando corrisponde al login o al logout. Per tutti gli altri comandi vedere la classe *CommandHandler.java*.
- *CommandHandler.java* : in questa classe vengono gestiti tutti i comandi per operare sui progetti, sulle carte, sui membri e sui vari spostamenti di essi. Per ogni comando viene eseguito il procedimento da effettuare per aggiornare le strutture dati. Alla fine di ogni aggiornamento della struttura dati viene mandata una notifica, quando necessario, a tutti i client registrati a quel servizio di notifica e viene mandata la risposta al client.
- *Registrazione.java* : al suo interno avviene la registrazione vera e propria dell'utente con aggiornamento della struttura dati e relativa notifica di tale aggiornamento. La notifica avviene sempre con il metodo RMI.
- *ServerNotification.java* : classe utilizzata per inviare effettivamente la notifica sullo stato degli utenti registrati a tutti i client registrati a tale servizio.
- *UpdateHashmap.java* : classe utilizzata per inviare effettivamente la notifica per l'avvenuto aggiornamento di qualche struttura dati a tutti i client registrati a tale servizio.
- *Progetto.java* : classe che rappresenta l'oggetto progetto. Esso è composto da un nome univoco e quattro liste che rappresentano il flusso di lavoro come passaggio delle card da una lista alla successiva. Le liste sono TODO, INPROGRESS, TOBEREVISED e DONE. Inoltre è presente un indirizzo IP e una porta UDP associati ad ogni progetto per la comunicazione nella chat.

- *Card.java* : classe che rappresenta l'oggetto card. Ad ogni card è assciato un nome, una descrizione e una lista di stringhe, dove ogni stringa rappresenta la storia dei vari spostami che la card ha fatto.

Client

Nella cartella client si trovano tutte quelle classi utilizzate per realizzare il client vero e proprio, per gestire tutti i comandi richiesti dall'utente, per gestire tutte le risposte arrivate dal server e per aggiornare le proprie strutture dati. Le classi presenti sono:

- *ClientMain.java* : classe utilizzata per la creazione delle strutture dati utilizzate dal client e per l'avvio del client vero e proprio.
- *ClientTCP* : corrisponde al client vero e proprio. Il client viene avviato attraverso la chiamata al metodo *startConnection()* nel *ClientMain.java*. All'interno di questa classe vengono effettutate tutte le registrazioni ai vari servizi di notifica, vengono gestiti tutti i comandi che l'utente digita da linea di comando per poter mandare al server il rispettivo messaggio di richiesta.
- *ClientNotification.java* : classe utilizzata per ricevere le notifiche sullo stato degli utenti registrati in maniera da poter aggiornare la propria struttara dati, *user_status*, che tiene traccia di tutti gli utenti registrati e del loro stato.
- *RemoteClient.java* : utilizzata per comunicare all'utente l'avvenuta registrazione o meno.
- *UpdateClient.java* : classe utilizzata per ricevere le notifiche sull'avvenuto aggiornamento di qualche struttura dati.

La cartella *worth* contiene anche altre quattro cartelle:

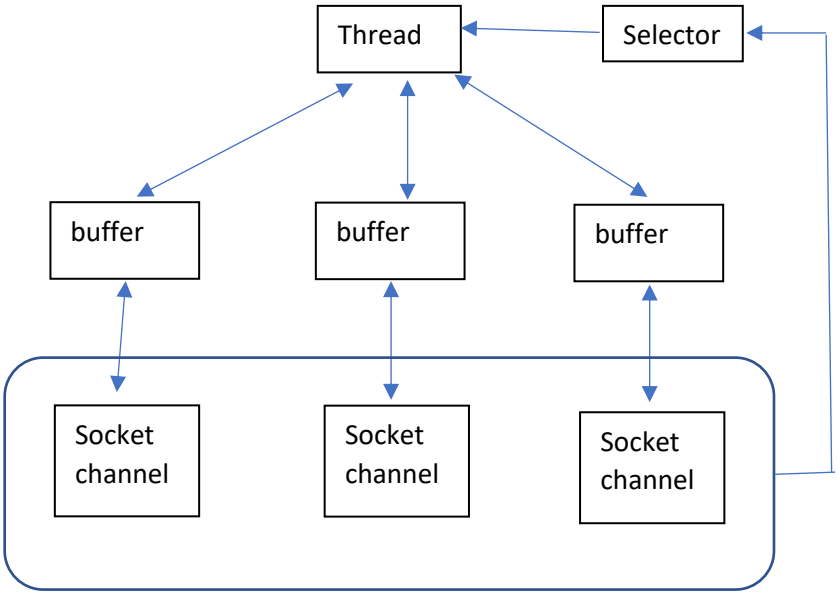
- *backupCards*
- *backupChat*
- *backupProjects*
- *backupUsers*

che rispettivamente conterranno il file per il ripristo dati delle carte, delle chat, dei progetti e degli utenti.

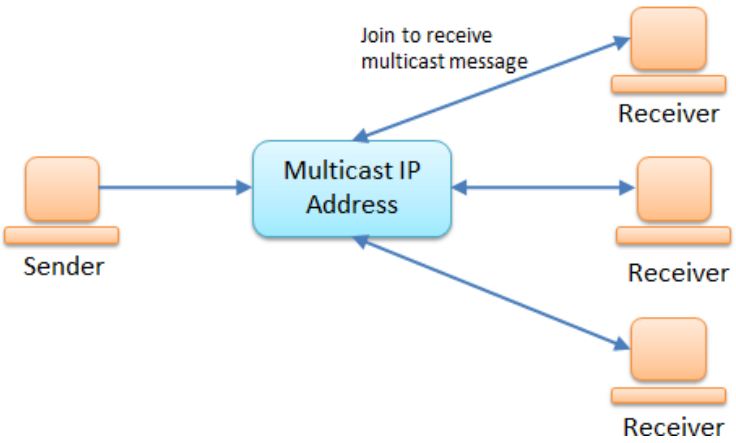
Scelte di implementazione

Per quanto riguarda la comunicazione client-server viene utilizzata una connessione TCP tramite multiplexing di canali TCP. La selezione dei canali avviene tramire il *Selector*. Il motivo di tale scelta risiede nell'aver bisogno della capacità di poter servire

un numero elevato di client che inviano richieste contemporaneamente. Tale bisogno è nato pensando al sistema WORTH come ad un sistema realizzato per essere utilizzato da chiunque e non solo per un fine accademico. La comunicazione è del tipo:



Per realizzare la comunicazione via chat viene utilizzata una connessione UDP. In particolare però, la chat non è altro che un thread che viene avviato ogni volta che un utente diventa membro dell'progetto relativo a quella chat. L'idea è quella di un thread che lavora in background, ovvero che riceva datagrammi, anche se il client è occupato nel gestire altro. La modalità con cui il server comunica ai client i riferimenti per unirsi alla chat è realizzata tramite notifiche realizzate con il sistema RMI. Ogni volta che un utente diventa membro di un progetto il server notifica tale evento al client passandogli la chat. Il client ricevuta notifica, e quindi la chat, la avvia tramite il metodo *.start()*. La connessione è del tipo:



La persistenza del sistema è stata realizzata tramite il salvataggio su file di oggetti di tipo *ObjectInputStream* e *ObjectOutputStream*. Ogni volta che le strutture dati sono

aggiornate vengono apportate le modifiche anche sui rispettivi file. Una volta che il server viene chiuso, alla successiva riapertura le strutture dati vengono ripristinate attraverso la chiamata ai rispettivi metodi di backup.