



Relazione per il progetto di “Laboratorio di Reti - Corso A”

A.A. 2020/2021

Tony Agosta 544090

**WORTH:
WORkTogetHer**

Indice

Introduzione	3
Visione generale del sistema e funzionamento	3
Istruzioni per la compilazione e avvio.....	4
Descrizione delle classi.....	5
Comunicazione Server-Client.....	7
Realizzazione del Server.....	7
Persistenza del Sistema.....	8

Introduzione

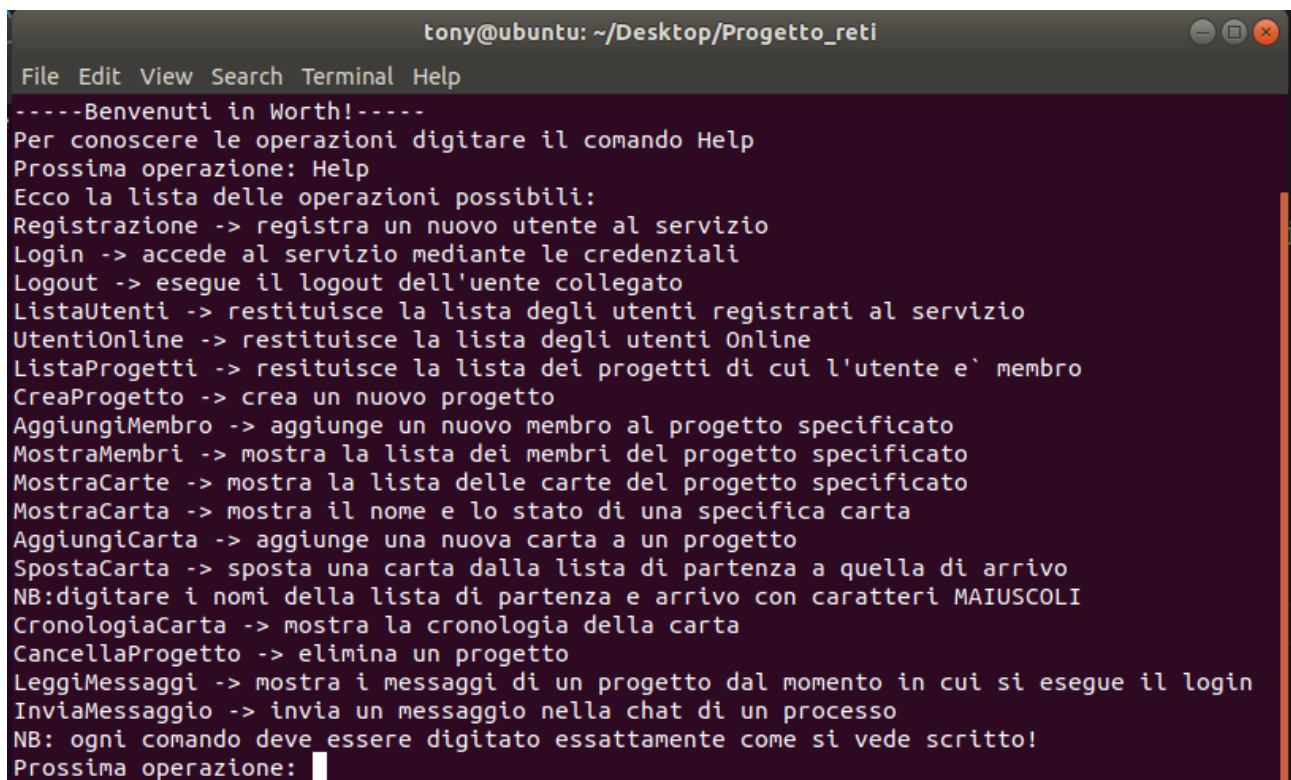
Il progetto ha lo scopo di realizzare una "piattaforma" per la gestione di progetti in modo da permettere a più persone di collaborarvi. In particolare, l'attenzione è stata posta sulla creazione e la gestione collaborativa di questi, sfruttando il metodo di gestione Agile che permette di avere una visione chiara dei compiti da svolgere per portare a termine un progetto.

Visione generale del sistema e funzionamento

Il sistema richiede la registrazione di un username con password prima di poter effettuare qualsiasi operazione. Dopo essersi registrati è possibile effettuare il Login per iniziare a interagire con la piattaforma.

L'operazione di registrazione è stata realizzata mediante il meccanismo Remote Method Invocation, la comunicazione tra client e server avviene tramite TCP su una apposita socket e, in fine, l'invio e la ricezione dei messaggi tra i membri di un progetto avviene tramite UDP.

All'avvio del client, digitando il comando "Help" sarà possibile vedere la sintassi (mostrata anche nella figura di seguito) dei comandi che possono essere digitati per interagire con il sistema.



```
tony@ubuntu: ~/Desktop/Progetto_reti
File Edit View Search Terminal Help
----Benvenuti in Worth!----
Per conoscere le operazioni digitare il comando Help
Prossima operazione: Help
Ecco la lista delle operazioni possibili:
Registrazione -> registra un nuovo utente al servizio
Login -> accede al servizio mediante le credenziali
Logout -> esegue il logout dell'utente collegato
ListaUtenti -> restituisce la lista degli utenti registrati al servizio
UtentiOnline -> restituisce la lista degli utenti Online
ListaProgetti -> restituisce la lista dei progetti di cui l'utente e' membro
CreaProgetto -> crea un nuovo progetto
AggiungiMembro -> aggiunge un nuovo membro al progetto specificato
MostraMembri -> mostra la lista dei membri del progetto specificato
MostraCarte -> mostra la lista delle carte del progetto specificato
MostraCarta -> mostra il nome e lo stato di una specifica carta
AggiungiCarta -> aggiunge una nuova carta a un progetto
SpostaCarta -> sposta una carta dalla lista di partenza a quella di arrivo
NB: digitare i nomi della lista di partenza e arrivo con caratteri MAIUSCOLI
CronologiaCarta -> mostra la cronologia della carta
CancellaProgetto -> elimina un progetto
LeggiMessaggi -> mostra i messaggi di un progetto dal momento in cui si esegue il login
InviaMessaggio -> invia un messaggio nella chat di un processo
NB: ogni comando deve essere digitato esattamente come si vede scritto!
Prossima operazione: 
```

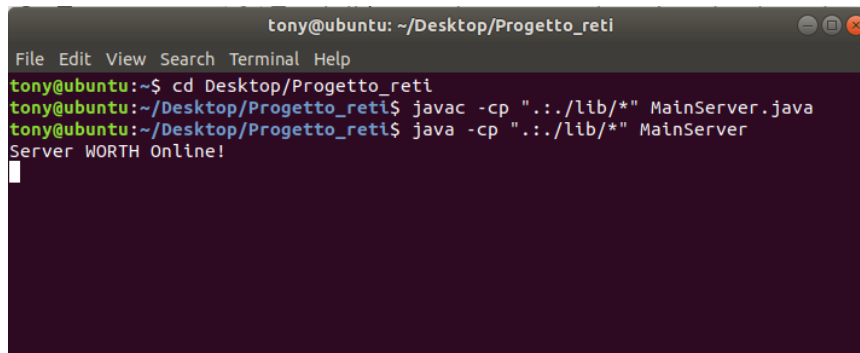
Istruzioni per la compilazione e avvio

Il progetto è stato testato su ambiente Linux. La cartella scaricata contiene anche directory "lib" che contiene le librerie esterne per il funzionamento di jackson utilizzato per la persistenza dei dati.

Server:

- Compilazione: innanzitutto bisogna spostarsi all'interno della directory che contiene la cartella "Progetto_Reti", dopodiché da terminale digitare il comando:
`javac -cp "../lib/*" MainServer.java.`
- Esecuzione: sempre rimanendo all'interno della directory digitare il comando:
`java -cp "../lib/*" MainServer`

Verrà mostrato un messaggio come in figura:

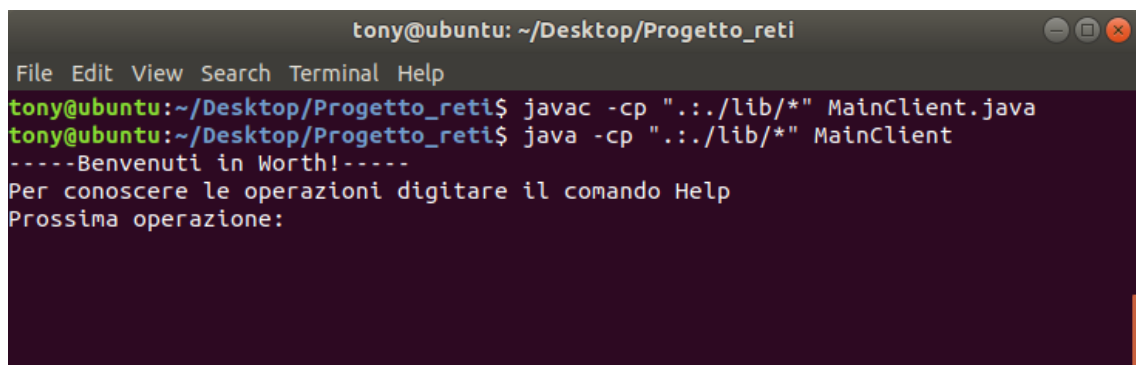


```
tony@ubuntu: ~/Desktop/Progetto_reti
File Edit View Search Terminal Help
tony@ubuntu:~$ cd Desktop/Progetto_reti
tony@ubuntu:~/Desktop/Progetto_reti$ javac -cp "../lib/*" MainServer.java
tony@ubuntu:~/Desktop/Progetto_reti$ java -cp "../lib/*" MainServer
Server WORTH Online!
```

Client:

- Compilazione: aprire un altro terminale, oltre a quello già aperto per il server, e, anche in questo caso, spostarsi all'interno della directory che contiene la cartella "Progetto_Reti", dopodiché digitare il comando:
`javac -cp "../lib/*" MainClient.java`
- Esecuzione: sempre rimanendo all'interno della directory digitare il comando:
`java -cp "../lib/*" MainClient`

Verrà mostrato un messaggio come in figura:



```
tony@ubuntu: ~/Desktop/Progetto_reti
File Edit View Search Terminal Help
tony@ubuntu:~/Desktop/Progetto_reti$ javac -cp "../lib/*" MainClient.java
tony@ubuntu:~/Desktop/Progetto_reti$ java -cp "../lib/*" MainClient
-----Benvenuti in Worth!-----
Per conoscere le operazioni digitare il comando Help
Prossima operazione:
```

A questo punto l'interazione con la piattaforma risulta molto facile e intuitivo!

Descrizione delle classi

MainServer: rappresenta la classe Main per il funzionamento del server.

Nel MainServer vengono creati i legami per gli oggetti che utilizzano i meccanismi di RMI per gli aggiornamenti delle strutture dati dei vari client connessi con il server. Inoltre, qui viene creato un oggetto di tipo Server che viene "attivato" invocando il metodo ".start()".

Server: nella classe Server si trova, oltre a tutti i metodi che implementano le funzionalità richieste, il metodo start() in cui viene creata e aperta la connessione TCP per permettere ai client di comunicare con esso.

Qui, vengono create e inizializzate le strutture dati che conterranno tutte le informazioni necessarie per il corretto funzionamento di tutto il sistema, i file e le directory che vengono utilizzate per rendere tutto il sistema persistente. La descrizione e il funzionamento dei metodi implementati in questa classe si trovano in "ServerInterface.java".

MainClient: rappresenta la classe Main per il funzionamento di ogni singolo Client, in cui viene creato un oggetto di tipo Client che viene "attivato" invocando il metodo ".start()".

Client: nella classe Client si trova il metodo start(), in cui viene creata e aperta la connessione con il Server, quindi se prima non avviene avviato il server non sarà possibile interagire con il sistema.

Card: la classe Card rappresenta una carta appartenente a un progetto ed è caratterizzata dai campi :

- o nameCard : nome della carta
- o description: descrizione della carta
- o history : cronologia dei movimenti
- o list: stato attuale della carta

Progetto: la classe Progetto rappresenta un singolo progetto ed è caratterizzato dai campi:

- o users: lista dei membri del progetto
- o projectName: nome del progetto
- o nameCards: lista dei nomi di tutte le carte del progetto
- o cards: lista di tutte le carte
- o TODO: lista delle carte che si trovano nello stato "TODO"
- o INPROGRESS: lista delle carte che si trovano nello stato "INPROGRESS"
- o TOBEREVIEWED: lista delle carte che si trovano nello stato "TOBEREVIEWED"
- o DONE: lista delle carte che si trovano nello stato "DONE"
- o ipAddress : è una stringa che viene utilizzata per le informazioni di connessioni relative alle Chat.¹
- o Project_port: è un intero che viene utilizzato per le informazioni di connessioni relative alle Chat.¹

¹ Per maggiori informazioni sulle connessioni si rimanda alla descrizione della classe ConnectionINFO

Utenti: la classe Utenti rappresenta un utente che si registra al servizio ed è caratterizzato dai campi:

- o nickname: nome dell'utente
- o password: password dell'utente
- o stato : stato dell'utente (online, offline)
- o ListaProgetti: lista dei progetti di cui l'utente è membro.

Utente_Stato: la classe Utente_Stato contiene le informazioni che vengono memorizzate nella struttura dati del client e che vengono aggiornate dal server tramite opportuni metodi che sfruttano il meccanismo RMI², ed è caratterizzata dai campi:

- o nomeutente: nome dell'utente
- o stato : stato dell'utente (online, offline)
- o project_list= lista dei progetti di cui l'utente è membro

Chat: la classe Chat è una classe di tipo Runnable che, viene mandata in esecuzione in background non appena viene creato un progetto (con la relativa chat) o quando un utente viene aggiunto ad un progetto in modo da permettergli di interagire con gli altri membri del progetto.

Gli oggetti di tipo Chat sono stati realizzati di tipo Runnable per sfruttare l'esecuzione continua di un thread senza dover bloccare le funzionalità dell'intero sistema. In questo modo un utente dal momento in cui effettua il login riceve i messaggi che vengono inviati nelle chat dei progetti di cui fa parte permettendogli di continuare a usare il sistema e senza dover stare in attesa dei messaggi.

Ma come funziona la Chat? Non appena l'utente effettua il login vengono attivati e quindi mandati in esecuzione tanti thread quanti sono i progetti di cui l'utente è membro, da questo momento verranno salvati tutti i messaggi che vengono inviati nelle chat dei progetti inclusi i messaggi relativi agli spostamenti delle carte. Se si vogliono leggere i messaggi di una chat di un particolare progetto basta digitare sul terminale il comando "LeggiMessaggi" e verranno mostrati i messaggi inviati nella chat del progetto specificato dal momento in cui è stato effettuato il Login. Attenzione però, se viene eseguito il comando per leggere i messaggi, da quel momento in poi sarà possibile solo leggere i messaggi ricevuti dopo l'esecuzione del comando "LeggiMessaggi". Similmente, per inviare un messaggio basta digitare sul terminale il comando "InviaMessaggio", specificare il nome del progetto dove inviare il messaggio e digitare il messaggio da inviare. I membri del progetto specificato riceveranno il messaggio inviato con il meccanismo descritto sopra.

Un oggetto di tipo chat è caratterizzato da :

- o address: l'InetAddress che viene utilizzato per le connessioni MulticastSocket
- o port : numero di porta uguale al progetto cui la chat si riferisce
- o nomeProgetto: nome del progetto a cui appartiene la chat
- o messaggiChat: lista dei messaggi ricevuti
- o multicastSocket : è il multicastSocket sfruttato per la connessione

² Per maggiori dettagli sui metodi che sfruttano RMI si rimanda al paragrafo "CallBack ed RMI"

ConnectionINFO: la classe ConnectionINFO è stata realizzata per contenere, in un unico oggetto, tutte le informazioni riguardo una connessione MulticastSocket. Ogni volta che viene creato un progetto, questo viene legato (tramite HashMap) a un oggetto di tipo ConnectionINFO che contiene le informazioni per creare e avviare la chat del progetto stesso.

Un oggetto di tipo ConnectionINFO è caratterizzato da:

- o multicastSocket : è il multicastSocket sfruttato per la connessione
- o address: stringa utilizzata per ottenere l'InetAddress per la connessione Multicast
- o port : numero di porta uguale al progetto cui la chat si riferisce
- o nomeProgetto: nome del progetto a cui appartiene la chat

RMI_Interface: RMI_Interface rappresenta l'interfaccia che contiene i metodi per mandare gli aggiornamenti agli utenti che si registrano al servizio di notifiche quando effettuano il login.

I metodi di questa interfaccia sono implementati nella classe Server.

NotifyInterface: NotifyInterface rappresenta l'interfaccia che contiene i metodi per aggiornare le strutture dati dei client.

I metodi di questa classe sono implementati nella classe Client.

RegisterCallback: RegisterCallback contiene i metodi che vengono utilizzati per aggiornare le strutture dati degli utenti che si sono registrati al servizio ma che non hanno ancora effettuato il login per la prima volta.

Comunicazione Server-Client

I Client e il Server comunicano mediante una comunicazione di tipo TCP per permettere di mandare dei messaggi dal client al server e viceversa. In particolare, il client invia al server sulla socket TCP un messaggio in cui indica il tipo di operazione che vuole eseguire e le informazioni necessarie per eseguire quell'operazione. Il server a sua volta risponde mandando un messaggio al client contenente l'esito dell'operazione.

Realizzazione del Server

Il server è stato implementato realizzando il multiplexing dei canali tramite NIO per consentire una interazione concorrente dei client con le risorse messe a disposizione dal server.

CallBack e RMI

Il meccanismo delle callback con RMI è stato ampiamente utilizzato per la realizzazione di tutto il sistema e per il corretto funzionamento dello stesso. In particolare, sono state definite due classi distinte che implementano e sfruttano i meccanismi delle CallBack tramite RMI.

Una classe, che manda notifiche per aggiornare le strutture dati degli utenti, che si sono registrati al servizio ma che non hanno ancora fatto il login per la prima volta, in modo da avere al momento del login, una struttura dati aggiornata che contiene le informazioni che lo riguardano. Ad esempio, se un utente si registra senza effettuare il login, può essere aggiunto a un progetto e alla relativa chat. Grazie a questo servizio di notifiche "offline" il nuovo utente una volta collegato potrà direttamente interagire con i progetti a cui è stato aggiunto.

Una seconda classe è stata realizzata per mandare notifiche di aggiornamento delle strutture dati degli utenti che hanno effettuato il login dopo la registrazione. Con questo servizio di notifiche "online" gli utenti possono aggiornare le proprie strutture dati in tempo reale.

Persistenza del sistema

La persistenza del sistema è stata realizzata aggiornando di volta in volta dei file e delle directory che vengono utilizzate per ripristinare tutte le strutture dati e le informazioni presenti nel sistema al momento della chiusura del server. In particolare, la serializzazione e la deserializzazione degli oggetti è stata realizzata utilizzando "jackson".