Relazione Progetto WORTH

Il progetto segue le specifiche date, utilizzando **un'interfaccia grafica** per il client, ma è comunque possibile usare un'interfaccia da terminale specificando la scelta al momento della run da terminale.
Per informazioni sulla compilazione e l'esecuzione, allego in **fondo** a questo documento le modalità.

Server
Il server è avviabile solo in modalità terminale, ma non è esclusa una possibile implementazione futura di un'interfaccia grafica.

Persistenza dei dati e lettura
All'avvio Inizializza attraverso il FileManager tutti i dati (in base al path definito)
Classe che utilizza la libreria jackson per serializzare e deserializzare file json. La struttura dei dati è la seguente :

Cartella "server" (è comunque possibile specificare un altro path in fase di esecuzione
se si desidera cambiarla)

- Cartella Projects (contiene tutti i progetti)
  - 1 file json per ogni progetto (specifica le informazioni relative a quel progetto, nome, creatore, membri)
  - 1 cartella per ogni progetto (contiene 1 file json per ogni card relativa al progetto, con nome, data di creazione e lista dei movimenti)
- Cartella Users (contiene tutti gli utenti)
  - 1 file json per ogni utente con nome, data di creazione dell'account e hash della password via md5.

Registra la risorsa DataBase alla porta *30001* all'indirizzo dell'host su cui viene eseguito (localhost) per richieste RMI.

- Viene creato un stub di tipo RMIWORTHServer alla porta 30001
- Il binding della stub viene effettuato attraverso il registro al nome "WORTH" , registro creato sulla porta 6789.
- Questa risorsa remota permette ai client di effettuare la registrazione, e ai client che effettuano il login di usare il meccanismo di callback per essere informati su eventuali login e logout da parte di altri utenti

Avvia un Thread TCPHandler che rende disponibile all'accettazione di richieste sullo stesso indirizzo, alla porta 9000. Il TCPHandler è implementato con un NIO Selector e SocketChannel, che permette la serializzazione delle richieste da parte dei client, per ogni connessione. Ogni nuova connessione ha associata un attachment che verrà poi utilizzato per "identificare" la sessione dell'utente e per comunicargli possibili esiti delle sue richieste.

Richieste gestite dal TCPHandler e dall'RMI
Se si effettua la connessione i client hanno la possibilità di effettuare due operazioni

- ***Login*** attraverso richiesta al **TCPHandler**
- ***Registrazione*** attraverso la risorsa remota con **RMI**
  La registrazione se va a buon fine consegue con un immediato login da parte dell'utente

    - Dopo aver effettuato il ***login*** (viene aggiornata la lista degli utenti online nel server) , il server offrirà al client la ***lista dei suoi progetti***, la ***lista degli utenti*** della rete (e quelli ***online***) attraverso il meccanismo di callback, e la possibilità di ***accedere o creare un progetto***.
        - Se la creazione va a buon fine ne consegue un diretto accesso al progetto appena creato (gestito lato client, cioè che se la richiesta di creazione va a buon fine il client richiede l'accesso al TCPHandler).
    - Una volta effettuato l'accesso il client potrà fare diverse operazioni legate al progetto
        - ***Ottenere l'indirizzo della chat relativa al progetto*** (viene effettuata una richiesta in "background" appena si accede a un progetto)
        - Ottenere la ***lista dei membri del progetto***
        - Ottenere la ***lista delle card relative al progetto***
        - ***Creare***, ***vedere***, ***muovere*** una card
        - Vedere lo ***storico dei movimenti relativi ad una card***
        - ***Aggiungere membri*** al progetto in questione
        - ***Eliminare il progetto*** ed anche ***effettuare il logout***

Gestione degli indirizzi multicast
Gli indirizzi multicast per le chat dei progetti vengono calcolati nel seguente modo:
Si prende l'indice di posizione del progetto nella lista dei progetti del server e si crea un indirizzo utilizzando questo indice.
es: indice = 456 b =(int)(indice/2^16) c = (int)((indice/2^8)%256) d = (indice%256)
indirizzo = 239.b.c.d → 239.0.1.200
La porta è uguale per tutti i progetti : 5000
P.S. 18/01/2021 se un progetto viene cancellato non viene gestito il riuso degli indirizzi, che avrei risolto utilizzando una lista di indirizzi non usati in cui aggiungevo gli indirizzi dei progetti cancellati in una sessione di avvio del server in modo da restituire questi prima di calcolare un nuovo indirizzo, se disponibili.

Ogni richiesta di modifica da parte del client di un progetto viene subito resa permanente richiedendo al FIleManager di scrivere (aggiornare) le informazioni relative a quel progetto, in modo da mantenere la persistenza.
Se un client si scollega senza prima aver effettuato il logout, verrà ovviamente lanciata un'eccezione che viene utilizzata per poter effettuare il logout di quell'utente dal server, altrimenti rimarrebbe online.

Client
Il client si avvia in modalità GUI di default (terminale se viene specificato in fase di esecuzione) , richiede la risorsa remota all'indirizzo dato via parametro (localhost se non viene passato alcun parametro) , e si collega al server TCP con SocketChannel, canale attraverso il quale poi effettuerà tutte le richieste.

- Per la registrazione utilizza il metodo remoto register(), se non va a buon fine viene informato con un dialog. Se va a buon fine viene immediatamente effettuato il login.
- Dopo aver effettuato il login, si registra alla lista dei client disponibili alla callback (RMI) utilizzando il metodo registerForCallback() dell'interfaccia RMIWORTHProject.

Effettuato il login si presentano le scelte effettuabili, sia da terminale che da GUI. La versione da terminale è più esplicativa dato che c'è un comando che permette di vedere le opzioni accessibili in quel momento ad un utente (-help).

1. Si possono vedere i propri progetti, gli utenti online, gli utenti della rete, e accedere o creare un progetto (creazione implica accesso al progetto, altrimenti non è andata a buon fine e viene informato l'utente con dei dialog)
    a. L'accesso a un progetto implica ottenere l'indirizzo multicast relativo alla chat, ovviamente il tutto effettuato in background.
2. Una volta fatto l'accesso ad un progetto si possono effettuare tutte le operazioni relative alle Card (Create, Show, Move, History), vedere la lista dei membri, vedere la lista delle card, eliminare il progetto (se possibile), inviare messaggi sulla chat, leggere la chat ed effettuare il logout.

Questo funzionamento è quindi "one-way" ovvero una volta effettuato l'accesso ad un progetto, per poter accedere ad altri progetti è necessario uscire e riavviare il programma.

P.S. Oggi *18/01/2021* ho valutato che questa mancanza può essere aggirato con circa 10 righe di codice, permettendo di accedere ai progetti semplicemente premendo il nome nella lista dei progetti dell'utente.

Strutture utilizzate nel Server
Si usano per lo più liste, restituite dalla Collection.synchronizedList().

- Per tutti gli accessi di tipo TCP l'accesso concorrente è garantito dall'utilizzo del Selector in TCPHandler, che gestisce una richiesta per volta, inefficiente se vogliamo essere puntigliosi, ma sicuramente adatto a questo particolare caso.
- Per le richieste di registrazione al server, registrazione e rimozione dalla lista dei client per la callback, si usa il blocco synchronized sul metodo in modo da utilizzare gestire l'accesso alla lista degli utenti, quindi solo un thread per volta potrà richiedere l'utilizzo del metodo. La lista degli utenti utilizza una struttura che è la conseguenza della chiamata di Collection.synchronizedList(), non garantisce l'atomicità della combinazione delle varie operazioni su di essa e ovviamente implica il blocco dell'intera struttura, ma semplifica il lavoro. Poteva essere utilizzata una Concurrent Collection, come una ConcurrentHashMap, che però in una ipotesi di utilizzo più orientata ai login che alle registrazione ritengo meno efficiente, dato che queste strutture danno il loro meglio in casi *more readings and fewer updates*.

Informazioni sulle entità in gioco

- ***Server***

  Il server è rappresentato da un insieme di più classi contenute tutte all'interno della cartella WorthServer (WORTH_Project/project/WorthServer).
  - ***DataBase***: La classe "principale" che gestisce tutti i dati è denominata DataBase (Implementazione di RMIWORTHServer, estende RemoteServer), in questa vi sono tutti i metodi per la *Registrazione*, *Login* e *Logout,* per *Creare*, *Accedere* o *Eliminare* un progetto, per *Creare*, *Richiedere*, *Spostare* e *Vedere lo storico* di una Card, in più vi sono presenti i vari metodi per *Richiedere* gli utenti della rete sia online che non, *Richiedere* la lista dei membri di un progetto, *Aggiungere* membri ad un progetto, *Richiedere* l'indirizzo della chat UDP Multicast del progetto. È presente un metodo "di servizio" per poter appunto comunicare ai membri dei progetti le relative modifiche a questi sulla chat UDP. Il metodo *register* come anche *registerforcallback* e *unregisterforcallback* sono implementazioni "necessarie" all'utilizzo di RMI: register permette di iscriversi al server, utilizzando la risorsa remota, *registerforcallback e il suo opposto* permettono ai client di mettersi nella lista dei client che vogliono ottenere informazioni al presentarsi di un evento (login o logout di altri utenti)
  - **RMIWORTHServer**: Interfaccia (che estende Remote) che definisce i metodi utilizzabili attraverso RMI.
  - **FileManager**: questo è il gestore dei file per la persistenza dei dati, ottiene dati per il DataBase e li aggiorna sempre per lui, usando *Jackson* e quindi json.
  - **TCPHandler**: Cuore del server per le connessioni da parte dei client, è una classe che estende Thread. Viene avviata assieme al DataBase, da lui stesso(DataBase), una volta inizializzati gli altri attributi. È un NIO Selector per permettere il collegamento e la gestione delle richieste dei client. (Selector perché? perché in questo modo l'accesso alle risorse è sequenziale)
  - **Project.Project**: Semplicemente una interfaccia che definisce il tipo Project. Viene implementata dalla classe ProjectImpl.
  - **Project.ProjectImpl**: Implementazione di Project, definisce i metodi e la struttura dei progetti. Abbiamo le 4 liste per le card, una lista di membri, un creatore, un nome di progetto e data di creazione.
  - **Project.Card**: Classe che definisce il tipo card, con nome descrizione e storico dei movimenti.
  - **User.User**: Interfaccia per il tipo User
  - **User.Account**: Classe che implementa il tipo User, con le varie informazioni dell'utente.
- ***Client***

  Il client è un insieme di classi presenti nella cartella omonima (WORTH_Project/project/Client)
  - **ClientInterface**: Interfaccia del tipo client (definita esclusivamente per il metodo di callback *notifyme*)
  - **Client**: Classe che rappresenta il client, contiene tutti i vari attributi, e i metodi per interagire con il Server, (in via empirica sono gli stessi che ha il server tranne per quelli RMI). Il client oltre al suo stesso Thread lancia un altro thread (indirettamente perché la classe ChatClient va ad avviare un Thread di tipo

MessageReader) che è quello della chat (**ChatClient**) per permettere la comunicazione con altri membri su un progetto. Può essere usato sia in via terminale che con l'interfaccia grafica.
    - **ChatClient**: Classe che gestisce le richieste di invio e ricezione di messaggi sull'indirizzo ip multicast dato dal server per il progetto in questione.
        - Lancia un Thread che è un Runnable (MessageReader) che resta in attesa all'indirizzo multicast della chat di progetto e riceve i messaggi del gruppo per poi scriverli sulla chat dell'interfaccia o sul terminale in modalità terminale.
    - **Frames.***: Sono tutti JFrame necessari all'implementazione delle varie interfacce grafiche, quali quella di login, accesso ai progetti, gestione dei progetti e chat.

Librerie Utilizzate

Jackson: Per la serializzazione e deserializzazione dei dati che per richiesta devono essere persistenti. **FlatLaf:** Per il lookandfeel dell'interfaccia grafica. **MyExceptions:** Semplice package contenente tutte le eccezioni lanciabili

Lato Server

L'implementazione DataBase utilizza 4 liste:

1. Lista dei progetti
2. Lista degli utenti
3. Lista degli utenti online
4. Lista dei progetti aperti

Inoltre è presente un'ulteriore lista che rappresenta i client registrati per la callback.
Fatta eccezione per la lista degli utenti e dei client che vengono accedute via RMI, le altre liste presenti sono utilizzate solo dal TCPHandler per soddisfare le richieste dei client, pur essendo tutte implementate come Synchronized Collections.

Il FileManager, utilizza jackson per serializzare e deserializzare i contenuti. Questo avviene attraverso richiesta specifica di diversi metodi quali getProjects e getUsers per i progetti e gli utenti, useranno poi updateProjects e updateUsers per le modifiche.

1. Il metodo getUsers non fa altro che leggere ogni file nella cartella degli utenti e ritornare una lista che appunto rappresenta gli utenti registrati fino a quel momento nel server
2. Il metodo getProjects oltre a leggere le informazioni relative al progetto stesso, ottiene anche tutte le card relative al progetto e le aggiunge al progetto in questione attraverso il metodo addCards() della classe Project.
3. Useranno poi updateProjects e

Project e Card, sono semplici classi che rappresentano le entità dei progetti e delle card dei progetti. Per permettere la persistenza delle card la classe project ha un metodo getCards() e un metodo addCards().

1. Il metodo getCards ritorna la lista delle card relative al progetto, utilizzato poi per la persistenza da parte del FileManager.
2. Il metodo addCards non fa altro che aggiungere tutte le card al progetto, con particolare attenzione nell'aggiungere la card alla lista appropriata ottenendo l'ultimo movimento legato a quella card.

RMIWORTHServer è semplicemente l'interfaccia necessaria alla definizione dei metodi remoti utilizzabili.

Lato Client
Client, è la classe principale che va a gestire la fruizione dei servizi offerti dal server. Alla sua creazione stabilisce una connessione TCP sull'indirizzo del server (localhost se su stesso host) alla porta 9000. E va ad ottenere la risorsa legata a "WORTH" attraverso il registro, sempre utilizzando lo stesso indirizzo ip ma questa volta alla porta 6789.

- All'utente verrà presentato un frame definito "loginFrame" che permette l'accesso o la registrazione. Dovrà inserire un nome utente e una password
    - Se uno dei due campi non è compilato verrà informato con un dialog di questa mancanza per invitarlo ad inserire i campi richiesti.
    - Se si verificano errori verrà informato con un dialog
        - con un messaggio "WrongPassword" se la password è errata
        - con un messaggio "NoUserWithThisName" se l'utente non esiste
        - con un messaggio "Utente già presente" se vuole registrarsi con un username già utilizzato
        - con un messaggio "password must be at least 4 characters long" se inserisce una password più corta di 4 caratteri.
- Dopo il login viene presentato un nuovo frame definito "MenuFrame", e chiuso quello di login, che permette di vedere attraverso due "menu" la lista dei progetti di cui l'utente fa parte, e la lista degli utenti sia offline che online (se online avranno una spunta verde).
    - Avrà un campo Project Name per inserire il nome del progetto che vuole creare o accedere
    - Se si verificano errori verrà informato con un dialog
        - con un messaggio "ProjectAlreadyExists" se vuole creare un progetto con un nome già utilizzato
        - con un messaggio "NotExistsProject" se vuole accedere ad un progetto non esistente
        - con un messaggio "UserNotMemberOfThisProject" se vuole accedere ad un progetto di cui non è membro

      (Come ho detto nelle prime pagine l'accesso ai progetti potrebbe essere molto più efficiente utilizzando il menu dei progetti come dei bottoni per accedere ai progetti)
- Il menu di un progetto definito "ProjectFrame" permette di effettuare tutte le operazioni legate ad un progetto quali
    - Formato da due frame uno che rappresenta le modifiche effettuabili al progetto quali creare card, vedere le informazioni relative ad una card, spostare una card da una lista ad un'altra, e vedere lo storico dei movimenti di una card.
        - Si può anche vedere la lista dei membri, la lista delle card, effettuare il logout ed eliminare il progetto.
        - Se si verificano errori il client viene informato attraverso un dialog
            - con un messaggio "empty" se uno dei cambi non è presente
            - con un messaggio "CardExists" se si vuole creare una card con lo stesso nome di un'altra "CardNotExists" se si vogliono effettuare operazioni su una card non esistente
            - con un messaggio "CantMoveCardFromTo" se si vuole spostare una card da un lista ad un'altra "saltando" alcuni passaggi

            - con un messaggio “CardNotIn” se la card che si vuole muovere non è nella lista specificata.
            - con un messaggio “ProjectNotFinished” se si vuole eliminare un progetto con card che sono ancora da terminare.
    - Il frame della chat è composto da 2 parti quella superiore che indica il progetto a cui è riferita e da la possibilità di aggiungere membri al progetto, l’altra che rappresenta l’area di testo su cui viene visualizzata la chat e l’area in cui scrivere i messaggi da inviare premendo il bottone.
        - Se si presentano errori nell’aggiunta di un utente al progetto viene informato l’utente con un dialog che descrive l’errore, che può essere in caso in cui l’utente è già membro del progetto o non esiste un utente con quel nome.

Tutti i frame sono creati attraverso la libreria Swing di java, che a mio parere rende complicato più del dovuto lo sviluppo delle funzioni.
Ogni azione compiuta sui vari frame è però solo un richiamo ai metodi della classe client in modo da permettere l’utilizzo di queste funzioni anche in via terminale, che ha lo stesso funzionamento in linea di massima ovviamente in via testuale.

Messaggi UDP

Server
I messaggi inviati dal server per informare delle varie operazioni non prevedono la join sul gruppo multicast, ma solo l’invio del messaggio alla coppia indirizzo porta che è stata assegnata al progetto. Quindi non viene effettuato il binding della MulticastSocket, ma solo l’apertura. L’indirizzo a cui mandare quindi sarà totalmente gestito dal DatagramPacket.

Client
Per la ricezione dei messaggi il client ovviamente va ad effettuare la join sulla chat multicast con una MulticastSocket. I messaggi ricevuti poi vengono salvati in una lista (locale) di messaggi se la chat è in modalità terminale per poterla poi leggere sotto richiesta, o semplicemente scritta sulla chat se in modalità GUI.
L’invio è gestito allo stesso modo in cui viene gestito sul server, con la differenza che il client ha fatto la join sul gruppo multicast. I messaggi inviati dall’utente corrente vengono modificati per simulare una vera chat sostituendo il nome del mittente con “you” nel caso in cui è l’utente stesso a leggere il messaggio.

Thread avviati da parte delle entità
Lato Server
- **MainThread**
- **TCPHandler**

Il server una volta creato un oggetto della classe DataBase va ad avviare un thread TCPHandler (che estende la classe Thread) dato che non avevo bisogno di estensioni ulteriori, questo si mettere in ascolto sulla socket “indirizzoip:9000” per accettare connessioni da parte di client.
Essendo il server avviato tramite una “classe” MainServer questo implica che verrà avviato un Thread main per questa classe, e questa lancerà a sua volta il thread TCPHandler.

Lato Client

- **MainThread (MainClient)**
- **MessageReceiver (ChatClient)**
- **EventDispatchThread (JFrame)**

All'avvio la classe client non esegue alcun thread ausiliario, che verrà avviato solo successivamente dopo l'accesso ad un progetto. Per ogni frame sono presenti i vari Thread (non gestiti direttamente dal programmatore) che vanno a catturare eventuali interazioni con i frame. Essendo la chat di tipo multicast non ho ritenuto necessario un thread per il continuo ascolto di messaggi da inviare. Come per il server ci sarà un thread relativo alla classe Main.

Per la compilazione e l'esecuzione porsi all'interno della cartella *Worth_Project/project/*

Se si presentano errori di compilazione o esecuzione copiando da questo pdf i comandi, verificare che non ci siano spazi anomali nel comando (si consiglia di copiarli prima in un blocco note)

Windows

Compilazione Windows

```
javac -cp MyLib/flat.jar;MyLib/core.jar;MyLib/data.jar;MyLib/anno.jar
*.java MyExceptions/*.java Client/*.java WorthServer/*.java
Client/Frames/*.java WorthServer/Project/*.java WorthServer/User/*.java
```

Server Execute

```
java -cp .;MyLib/flat.jar;MyLib/core.jar;MyLib/data.jar;MyLib/anno.jar
MainServer [facoltativo path per i file]
```

Client Execute

```
java -cp .;MyLib/flat.jar;MyLib/core.jar;MyLib/data.jar;MyLib/anno.jar
MainClient [IP server(default localhost)] [tipo interfaccia
{terminal , gui}]
```

UNIX

Compilazione UNIX

```
javac -cp MyLib/\* *.java MyExceptions/*.java Client/*.java
WorthServer/*.java Client/Frames/*.java WorthServer/Project/*.java
WorthServer/User/*.java
```

Server Execute

```
java -cp .:MyLib/core.jar:MyLib/data.jar:MyLib/anno.jar:MyLib/flat.jar
MainServer [facoltativo path per i file]
```

Client Execute

```
java -cp .:MyLib/core.jar:MyLib/data.jar:MyLib/anno.jar:MyLib/flat.jar
MainClient [IP server(default localhost)] [tipo interfaccia
{terminal , gui}]
```