Descrizione Progetto:
Server:
WorthServer ha un'architettura basata su un main loop che funziona da command parser, il quale riceve comandi sotto forma di stringhe inviate dai vari client con una connessione tcp che effettua multiplexing e demultiplexing.
Il main loop (contenuto nella classe CommandServer) attiva una ServerSocketChannel non bloccante in ascolto per default sulla porta 6001.
Per il servizio di registrazione a WORTH, viene utilizzata la classe RegistrationService, che implementa RegistrationServiceInterface.
RegistrationService utilizza una hashmap per memorizzare il nome utente e la relativa password per ogni utente registrato a WORTH. I metodi di questa classe permettono la registrazione ed il login da parte di un utente. Per il login viene usata la comunicazione client-server sul canale descritto in CommandServer, mentre per quanto riguarda la registrazione viene usato l'oggetto remoto descritto da RegistrationServiceInterface (per questo RegistrationService estende la classe RemoteServer). RegistrationServiceInterface infatti implementa Remote, e dichiara il solo metodo register, che può essere chiamato dal client sull'oggetto remoto per registrarsi a WORTH.
La classe UserListCallback, che implementa UserListCallbackInterface, si occupa invece di mantenere aggiornata la lista degli utenti con il relativo status (online oppure offline) utilizzando ancora una HashMap, con campo chiave relativo ai nomi utente e campo valore booleano, che rappresenta lo status. Questa classe si occupa poi di notificare ogni client connesso a WORTH di eventuali cambiamenti nello status degli altri utenti. Permette quindi ai client, come specificato dalla classe UserListCallbackInterface, di registrare lo stub del proprio oggetto remoto per ricevere le notifiche riguardanti la lista utenti. Tutto questo è implementato come nel caso di RegistrationService con la tecnologia RMI.
RegistrationService e UserListCallback sono connessi tramite un riferimento interno a RegistrationService, così da poter aggiornare UserListCallback ad ogni nuovo login o registrazione.
Il registro con il quale è possibile ricavare i riferimenti ai vari oggetti remoti è di default sulla porta 6000.
Il metodo main si occupa di controllare se le directory ed i files che servono a permanere lo stato del sistema su disco esistono, e in caso contrario le crea. Successivamente inizializza il servizio di notifica UserListCallback, recuperando le informazioni sugli utenti registrati (se esistono). In fine inizializza RegistrationService, anche qui recuperando le informazioni da filesystem, per poi lanciare il ciclo principale con la classe CommandServer.
Per modificare le porte associate ai due canali di comunicazione del server è necessario avviare l'esecuzione con due argomenti, rispettivamente la porta per la comunicazione TCP di CommandServer e la porta per il registro di localizzazione degli oggetti remoti.
Gli oggetti remoti stessi vengono attivati su porte libere casuali, avendo indicato nei relativi metodi exportObject la porta numero 0.
La presenza di eventuali thread attivati dagli oggetti remoti non è trasparente al programmatore, per questo motivo tutti i metodi contenuti nelle classi che fanno uso di RMI, cioè RegistrationService e UserListCallback, sono dichiarati synchornized. Così facendo si evita l'eventualità di deadlock e race conditions.
WorthServer non implementa altri meccanismi multithread, poiché la gestione del ciclo principale in CommandServer utilizza un canale di comunicazione che effettua multiplexing/demultiplexing. I messaggi in entrata sul canale non possono superare la dimensione di 256 bytes, e questo è garantito dal client. Quindi non appena il server nota l'intenzione del client di inviare un messaggio, il primo può leggere tutto il messaggio su l'unico buffer utilizzato da CommandServer, di dimensione 256 bytes, e poi procedere al parsing del comando (utilizzando un tokenizer di stringhe) e all'esecuzione dello stesso.
Per quanto riguarda la risposta invece, non si ha garanzia che il messaggio sarà di dimensioni inferiori a 256 bytes, e quindi CommandServer provvede a spezzare il messaggio aggiungendo un carattere speciale '$', che segnalerà al client la necessità di leggere anche la parte successiva. Il

server quindi, dopo aver spezzato il messaggio, segnalerà ancora l'intenzione scrittura verso il client.
Il carattere speciale usato per tokenizzare le stringhe e distinguere il comando ed i suoi argomenti è il carattere "|". Il client ci garantisce che questo carattere non verrà mai inviato come contenuto di un comando o di un argomento.
La classe Project è l'implementazione di un progetto WORTH. Essa contiene diversi TreeSet, uno per la lista dei membri, le altre contengono le cards. Questi set hanno infatti il nome delle diverse liste di un progetto, ovver "todo", "inprogress", "toberevised" e "done".
La classe Card invece contiene il nome della carta, la sua descrizione e una lista, contenente la history degli spostamenti della card.
Ogni qual volta si opera su un progetto, i cambiamenti vengono registrati sul filesystem, nella cartella ./res/projects/-nome del progetto-. Questa cartella contiene un file per ogni card -nome della card-.txt e il file "members", contenente il nome di ogni membro del progetto.
La struttura testuale con la quale sono scritti i files delle card è la seguente:
nome della card|descrizione|todo| -altre liste nelle quali la card ha transitato, separate da |
Così facendo la scrittura dei dati riguardanti ogni spostamento della card può essere fatta senza dover sovrascrivere l'intero file, ma semplicemente aggiungendo in fondo al file il nome della lista seguito da |. Per implementare la scrittura su filesystem sono state usate funzioni della libreria java.IO.
Per recuperare le informazioni sulle card e sui progetti all'avvio, il server si limita a scandagliare tutte le cartelle all'interno della cartella ./res/projects, per le quali si crea un progetto con il nome della cartella, e si recuperano i rispettivi membri e le card con i files all'interno di quest'ultima utilizzando un tokenizer per il carattere "|".
Ogni progetto ha associato un indirizzo ip multicast che sarà fornito ai client che vogliono unirsi alla chat di tale progetto col comando JoinChat. L'indirizzo è unico (uno per progetto) e viene deciso al momento della creazione del progetto (quindi cambia ogni volta che il server viene riavviato e quindi ricrea l'istanza di ogni progetto) utilizzando la classe MulticastAddressSpace. MulticastAddressSpace è una classe astratta che si occupa solamente di assegnare indirizzi IP multicast per le chat di progetto. Questa contiene il solo metodo getAddress, che fornisce un indirizzo IP ed incrementa il contatore interno alla classe per fornire un nuovo indirizzo IP al prossimo chiamante della funzione. Il metodo non è dichiarato synchronized perchè come già specificato (e questo vale anche per la classe Project e Card) il server opera nel suo ciclo principale in modo single thread. Quindi non possono verificarsi meccanismi di concorrenza per tutti i metodi e i dati che non riguardano le classi UserListCallback e RegistrationService.
Se si desidera cambiare le porte di WorthServer, bisogna lanciare il programma con due argomenti, uno per la porta relativa alla comunicazione TCP nel ciclo principale, e l'altra relativa al registro di localizzazione degli oggetti remoti.

Per la serializzazione ed il mantenimento dei dati persistenti su FileSystem non sono state usate librerie di serializzazione esplicita, ma bensì scritture su file testuali di elementi separati dal carattere "|" usando la libreria java.IO. Questa scelta deriva dalla semplicità di poter aggiungere informazioni in coda al documento senza doverlo sovrascrivere interamente ogni volta. Ad esempio, se una Card viene spostata da una lista a un altra, è sufficiente aggiungere in coda al file il nome della lista seguito da "|". Usando una libreria di serializzazione come jackson, invece, si sarebbe dovuto riscrivere l'intero file con la stringa equivalente alla Card serializzata, oppure agire in modo tale da separare la serializzazione della history da quella della Card. Le operazioni di serializzazioni quindi sarebbero risultate più costose per il server.

Client:
WorthClient agisce come command line parser non appena va in esecuzione il suo main loop, all'interno della classe CommandClient. Qui vengono letti input su System.in ed eseguiti i relativi comandi. Per ogni nome, che sia esso di progetto, utente, o nome di una card, è stato imposto il vincolo che non possano superare i 12 bytes. Per le descrizioni delle card è invece imposto il limite di lunghezza di 32 bytes. Sono inoltre vietati i caratteri speciali ovunque tranne che nelle descrizioni delle card e nei messaggi della chat (è comunque proibito l'uso dei caratteri "|" e "$" nelle descrizioni di card). Qualora uno di questi caratteri è inserito in una linea di comando, il client provvederà a lanciare un messaggio di errore.
I messaggi inviati al server con la comunicazione TCP da CommandClient non possono superare la dimensione del buffer, ovvero 256 bytes. Quelli ricevuti invece possono superare tale dimensione, è quindi prevista la lettura di messaggi in più fasi, separati dal carattere "$" (come descritto lato server).
CommandClient mantiene il nome dell'utente loggato per tutta la sessione, impedendo l'utilizzo di comandi insensati, come il tentativo di effettuare un'altra login mentre o registrazione mentre si è già loggati, e vice versa, impedisce l'uso di comandi che non siano quelli per la registrazione ed il login fino a che non si è acceduti.
Per la registrazione è usata la tecnologia RMI, chiamando il metodo register specificato in RegistrationServiceInterface. Se questa va a buon fine, il client si occupa di creare uno stub del proprio UserListRemote da inviare al server per i callback delle notifiche su eventuali modifiche alla lista utenti usando il metodo registerForCallback dell'interfacca UserListCallbackInterface, che restituisce la map aggiornata di tutti gli utenti al momento del login.
Il servizio di LocateRegistry per localizzare gli oggetti remoti pubblicati dal server è per default sulla porta 6000.
Per garantire la safety dell'accesso asincrono a UserListRemote (usato dal server per notificare i cambiamenti alla lista utenti, ma che contiene anche la medesima) tutti i metodi della classe sono dichiarati synchronized.
La gestione di tutti i comandi per operare su progetti è ovviamente delegata al server, il client si occupa solamente di controllare che siano inviati il giusto numero di argomenti, e che non vi siano caratteri speciali.
Le chat di progetto alle quali l'utente si è unito sono memorizzate in una lista all'interno di CommandClient. Per accedervi si usa il comando JoinChat [nome del progetto]. In risposta a questo comando, il server ci fornisce l'indirizzo ip multicast unico associato alla chat del progetto.
La classe Chat è una classe che estende Thread. Per ogni progetto per il quale si vuole partecipare alla chat, viene infatti avviato un nuovo thread che esegue il metodo run della classe chat.
All'interno del metodo run, il thread si mette in attesa della ricezione di un messaggio inviato da altri membri della chat con UDP. Per scrivere un messaggio, invece, si accede al metodo send dall'esterno.
Ogni qual volta il thread della classe Chat riceve un messaggio, questo viene inserito in una lista, che serve all'utente per leggere la totalità dei messaggi ricevuti sulla chat di progetto dall'ultima chiamata del metodo readChat. Ad ogni invocazione di readChat la lista dei messaggi ricevuti viene stampata a schermo e poi svuotata. L'accesso alla lista dei messaggi è controllato da un meccanismo di lock, per impedire che venga effettuata un'operazione di lettura e di scrittura in contemporanea, che renderebbe nulla l'integrità dei messaggi.
La ogni chat di progetto è contraddistinta dal proprio indirizzo IP multicast, poiché la porta è sempre la medesima (7001 per default).
Per lanciare WorthClient con porte differenti da quelle di default, bisogna lanciare l'esecuzione del programma con tre argomenti, rispettivamente: porta per la comunicazionet TCP, porta per il registro di localizzazione di oggetti remoti, porta per la comunicazione UDP della chat.

Comandi e sintassi lato Client:

per la registrazione:
Register/register nome_utente password

per il login:
Login/login nome_utente password

per vedere la lista degli utenti registrati:
ListUsers/listUsers/listusers

per vedere la lista degli utenti online al momento:
ListOnlineUsers/listOnlineUsers/listonlineusers

per sloggare e continuare ad usare WORTH:
Logout/logout

per sloggare e chiudere WORTH:
Quit/quit

per creare un progetto:
CreateProject/createProject/createproject nome_progetto

per aggiungere un membro al progetto:
AddMember/addMember/addmember nome_progetto nome_utente_da_aggiungere

per aggiungere una card al progetto:
AddCard/addCard/addcard nome_progetto nome_card descrizione

per spostare una card da una lista all'altra all'interno del progetto:
MoveCard/moveCard/movecard nome_progetto nome_card lista_di_partenza lista_di_arrivo
(i nomi delle liste sono tutti in minuscolo e senza spazi)
esempio: movecard progetto1 carta3 todo inprogress

per vedere i membri di un progetto:
ShowMembers/showMembers/showmembers nome_progetto

per vedere i dettagli di una carta (nome, descrizione e lista di appartenenza):
ShowCard/showCard/showcard nome_progetto nome_card

per vedere tutte le carte di un progetto nelle relative liste:
ShowCards/showCards/showcards nome_progetto

per vedere gli spostamenti di una carta:
GetCardHistory/getCardHistory/getcardhistory nome_progetto nome_card

per chiudere un progetto (tutte le card devono essere in done):
CancelProject/cancelProject/cancelproject nome_progetto

per unirsi alla chat di un progetto:
JoinChat/joinChat/joinchat nome_progetto

per inviare un messaggio sulla chat di un progetto:
SendChatMsg/sendChatMsg/sendchatmsg nome_progetto messaggio

per leggere la chat di un progetto:
ReadChat/readChat/readchat nome_progetto

per lasciare la chat di un progetto:
LeaveChat/leaveChat/leavechat nome_progetto

per la lista dei progetti di cui si è membri:
ListProjects/listProjects/listprojects

Istruzioni per importazione ed esecuzione:
Il progetto può essere importato su eclipse utilizzando “import an existing java project” su entrambi i file archivio WORTHClient.zip e WORTHServer.zip
Non sono state usate librerie esterne, quindi non dovrebbero presentarsi problemi con il buildpath.
Al primo avvio del server verranno inizializzate le cartelle ed i files necessari per il mantenimento delle informazioni persistenti su filesystem, all'interno della cartella ./res nella directory del progetto WORTHServer.
Per il corretto funzionamento bisogna lanciare per primo WORTHServer, e poi un numero a piacere di istanze di WORTHClient. Qualora si ottenga un eccezione di binding, bisogna lanciare sia server che client con altre port come parametri.
Ad esempio:
Parametri lato Server: 10000 10001
In questo caso il server inizalizzarà la ServerSocketChannel del ciclo principale sulla porta 10000 e il LocateRegistry sulla porta 10001.
Parametri lato client: 10000 10001 10002
Così che il client connetterà la sua SocketChannel alla ServerSocketChannel sulla porta 10000, utilizzerà il LocateRegistry sulla porta 10001 e si connetterà alle chat sulla porta 10002.