Relazione progetto di Laboratorio di Reti Dei Calcolatori

A.A. 2016/2017
Lisa Cacciamano

Il progetto consiste nello sviluppo di un gioco multiplayer, in cui vengono inviate al client delle lettere e l'utente deve inserire più parole possibili con tali lettere prima dello scadere del tempo.

Alcune classi sono usate sia dal client che dal server, in particolare **SocketTCP** e **SocketUDP** le quali mettono a disposizione i metodi per la creazione e la gestione delle socket rispettivamente TCP E UDP.
SocketTCP non include la socket server.

Sia il server che il client usano il file "Costanti.ini" della carella "file" per l'inizializzazione, quindi gli eseguibili "TwistGameClient.jar" e "TwistGameServer.jar" si devono trovare nella stessa cartella della cartella "file".

Server

Il server viene avviato tramite terminale con i con i comandi:
javaw rmiregistry
java -jar TwistGameServer.jar
Appena avviato mostra la porta e l'IP su cui è in ascolto in attesa di richieste da parte dei client.

Il metodo main si trova nella classe **TwistServerMain**. Questa inizializza i dati necessari al server, avvia il server RMI per il login/logout e la registrazione e avvia il thread per l'attesa delle richieste dei client in TCP.

DatiTwistGame raggruppa i dati necessari al funzionamento del server.

- public Map<String, Utente> utentiRegistrati;
  Insieme degli utenti registrati. La classe Utente fornisce i metodi statici per serializzare e deserializzare utentiRegistrati.
  Quando viene avviato il server viene creata una Map contenente i dati serializzati nel file FILE_UTENTI, oppure viene creata una Map vuota se il file non esiste. Gli utenti vengono salvati in una Map in modo da rendere la ricerca più veloce. La struttura è resa thread safe tramite la sincronizzazione (Collections.synchronizedMap(mp)). L'accesso il scrittura e lettura al file contenete i dati degli utenti e reso sicuro tramite synchronize(FILE_UTENTI).
  La classe Utente mette a disposizione anche i metodi per memorizzare e visualizzare il punteggio generale, per verificare la password e per stabilire se un utente è online o no.
  Ogni accesso ai dati dell'utente è synchronize.
  L'istanza viene usata viene usata per salvare lo stub RMI del client una volta che ha effettuato il login.

- public Dizionario dizionario;
  Oggetto contenente il dizionario (insieme delle parole consentite dal gioco) e i metodi per gestirlo. Le parole sono salvate in un Set per rendere la ricerca veloce.

- public Costanti costanti; Inizializza le costanti usate dal server, quali:
  - l'IP dell'host su cui si trova il server (IP_SERVER)
  - l'IP per il server multi cast (IP_MULTICAST)
  - la porta del registro RMI (REGISTRY_PORT)
  - la porta iniziale per le connessioni (PORTA_INIT)
  - la cartella dei file (PATH),
  - il nome del file contenete il salvataggio degli utenti registrati (FILE_UTENTI),
  - il nome del file contenete il dizionario (DICTIONARY)

  Inoltre contiene le costanti per la comunicazione tra client e server.
  Le costanti IP_MULTICAST, REGISTRY_PORT, PORTA_INIT, PATH, FILE_UTENTI, DICTIONARY sono inizializzate tramite il file "Costanti.ini".
  Per le costanti inizializzate vengono messi a disposizione i metodi get per impedirne la modifica.
  In questa classe sono presenti anche i metodi per ottenere le porte relative ad una specifica partita.

- public AtomicInteger numPartite; Inizializzato ad uno, tiene traccia del numero di partite che il server ha creato. Viene usato, insieme al PORTA_INIT, per trovare su quale porta il client e il server comunicano durante la partita; per questo viene incrementato di due ogni volta che un utente richiede una nuova partita (ogni partita necessita di una porta su cui comunicare con TCP e UDP, e una porta per il multicast). Viene usata una variabile atomica per renderla thread safe poiché l'incremento avviene nel thread che gestisce la partita.

TwistServerTCPaccept implementa un thread che rimane in attesa delle connessioni dei client sulla porta PORTA_INIT. La connessione prevede l'autenticazione del client tramite username, si assume che il cliente abbia già effettuato il login. Una volta identificato l'utente avvia il thread TwistServerHandler per la gestione delle richieste.

TwistServerHandler per prima cosa attende la richiesta dal client: gestisce le richieste di creazione di una nuova partita o di visualizzazione della classifica generale. La classe contiene i metodi:

- private boolean nuovaPartita()

Crea una nuova istanza Partita(id, dizionario); l'id viene generato usando la variabile atomica numPartite, che viene anche incrementata di 2. L'istanza contiene tutte le informazioni che saranno necessarie alla gestione della partita stessa.
A questo punto il server attende la lista degli utenti invitati alla partita, verifica che gli utenti siano registrati e online (non consente di invitare se stesso), e se non si verificano problemi aggiunge gli utenti alla lista degli utenti invitati nell'istanza partita.
Una volta eseguiti i controlli inoltra gli inviti tramite callback RMI e conferma l'inizializzazione della partita al client che l'ha richiesta, prima di inviare l'invito anche a lui. Nell'invito è presente anche l'id della partita che servirà all'utente per sapere a quale porta collegarsi.

- private boolean setupNuovaPartita()

Se l'inizializzazione della nuova partita è avvenuta con successo viene chiamato questo metodo.
Apre una ServerSocket e setta il timeout per la receive a 7 min. A questo punto attende le connessioni dei client per la conferma o il rifiuto dell'invito. Una volta che il cliente si è

autentificato tramite username apre un thread a cui viene passato anche la socketServer (**SetupPartitaHandler**).
Questo thread riceverà il messaggio del client e nel caso sia di conferma creerà un nuovo partecipante nell'istanza partita e vi memorizzerà anche la socket tramite cui invierà le lettere della partita o il messaggio di cancellazione partita; in caso tutti gli invitati abbiano accettato l'invito o un utente abbia rifiutato la partita, vine chiusa la serverSocket per interrompere il setup della partita.
A questo punto il thread principale verifica che la partita sia valida: in caso non tutti gli utenti abbiano accettato l'invito viene inviato il messaggio di cancellazione partita a coloro che avevano accettato.

- private void avvioPartita()

Quando la partita viene avviata viene avviato il thread per il multicast (**TwistServerMulticast**) tramite cui verranno inviati i risultati. I thread comunicheranno tramite una BlockingQueue. Il multicast viene avviato a questo punto per permettere a tutti i client di registrarsi ad esso prima dell'inizio dell'invio dei risultati.
A questo punto vengono settate le lettere e inviate ai client. La classe Lettere contiene tutti i metodi per gestire le lettere e successivamente permettere di calcolare il punteggio delle parole.
Il thread rimane in attesa dei dati dai client tramite una socket UDP con timeout settato a 5 minuti. L'attesa si interrompe se tutti i partecipanti hanno inviato i risultati.
È previsto che il client invii una stringa contenente le parole separate da "_" e precedute dal nome utente, anche esso separato da "_".

- private void risultatiPartita()

Questo metodo elimina le parole doppie inserendole un Set per poi richiamare il metodo calcoloPunteggio (utente, listaParole) dell'istanza partita. Questa userà il metodo di Lettere per calcolare il punteggio di ogni parola e lo aggiungerà al punteggi della partita dell'utente corrente.
A questo punto vengono inviati i risultati ai client che si sono registrati al server multicast e aggiornati i punteggi generali in utenti Registrati in modo che poi possano essere serializzati.
Il punteggio di ogni parola viene calcolato così:
- viene verificato che la parola sia presente nel dizionario.
- la parola da testare viene trasformata in un array di caratteri.
- i punti di ogni parola equivalgono al numero delle lettere
- vengono tolte dalla parola da testare tutte le lettere della partita
- se alla fine la parola da testare contiene ancora lettere le viene assegnato 0 punti in quanto non è valida, altrimenti gli vengono assegnati i punti calcolati in precedenza.

- private void visualizzaClassifica()

Preleva i punteggi generali da utentiRegistrati, crea una classifica, la ordina secondo il nome dell'utente e la invia al client.

TwistServerRMI implementa i metodi messi a disposizione dall'interfaccia TwistServerInterface.

- Login: setta lo stato di un utente registrato ad attivo.
- Logout: setta lo stato di un utente registrato a non attivo.
- Registrazione: crea un nuovo utente e lo aggiunge alla lista degli utentiRegistrati, prima di serializzarla.

Client

Per avviare il client è sufficiente aprire l'eseguibile "TwistGameClient.jar"
Il client ha un interfaccia grafica che mette a disposizione le operazioni di login, logout, registrazione, nuova partita, classifica generale e visualizza inviti, oltre ai campi per l'inserimento di username e password. Alcune operazioni vengono rese disponibili solo dopo aver effettuato il login.

Il bottone nuovaPartita apre una nuova finestra di dialogo che permette di inserire gli avversari tramite il pulsante aggiungi o il tasto invio. Una finestra simile viene usata per permettere di inserire le parole durante il gioco.

La classe astratta **TwistClientGUI** implementa l'interfaccia grafica e tutti i metodi necessari alla sua gestione, gli event listener invece sono creati nel main.
InputDialog è una classe astratta che implementa le finestre di dialogo per gestire l'input dell'utente. Le creazioni di tali finestre avviene nelle classi **TakeParole** e **TakeAvversari**, mentre la gestione degli eventi di tali finestre è implementata in **GestioneInputDialog**. La classe **GestioneClientGUI** gestisce gli eventi dell'interfaccia principale.
Le operazioni di login, logout e registrazione vengono eseguite direttamente nel thread dell'interfaccia essendo brevi, mentre le operazioni di nuova partita, setup partita e visualizza classifica vengono eseguite in thread diversi per evitare di bloccare l'interfaccia.

Il pulsante Exit e la chiusura della finestra tramite la x hanno lo stesso effetto: viene effettuato il logout e poi viene chiuso il client.

Il metodo setDefaultFontSize() inizializza la dimensione dei font utilizzati nell'interfaccia a seconda della risoluzione dello schermo, in quanto si era presentato il problema che l'interfaccia non era compatibile con lo schermo UHD del mio computer, le scritte erano troppo piccole per poter essere lette.

Per il passaggio dell'input da un thread all'altro viene usata un istanza della classe ListString che permette di memorizzare l'input inserito, verificare se è stato inserito qualcosa e permette al thread che dovrà usare l'input di recuperare l'array contenente le stringhe.

Il metodo main si trova in **TwistClientMain.** Qui viene creata l'interfaccia utente, l'istanza TwistClientRMI e rimane in ascolto degli eventi dell'interfaccia. Inoltre vengono anche inizializzate le costanti tramite il file "Costanti.ini"

- IP_SERVER: IP del server a cui si deve collegare
- IP_MULTICAST: IP del server multi cast al quale deve collegarsi per ricevere i risultati della partita
- REGISTRY_PORT: porta del registro RMI
- PORTA_INIT: porta del server usata per le richieste TCP di nuova partita e visualizza classifica, viene inoltre utilizzata insieme all'id partita per sapere su quali porta viene gestita la partita corrente.

Per tali costanti sono presenti i metodi get per impedirne la modifica in momenti successivi all'inizializzazione.
Le costanti utilizzate per la comunicazione tra client e server non vengono inizializzate tramite il file "Costanti.ini" e essendo final non necessitano dei metodi get.
In questa classe sono presenti anche i metodi per ottenere le porte relative ad una specifica partita.

TwistClientRMI gestisce il client RMI e le relative operazioni quali login, logout e registrazione.

NuovaPartita consente di inizializzare una nuova partita. Il client chiede all'utente di inserire uno o più avversari da invitare; se la finestra viene chiusa senza inserire avversari viene annullata l'operazione visualizzando un messaggio che deve essere inserito almeno un avversario.
Viene aperta una connessione TCP con il server tramite la quale viene inviato username, la richiesta di una nuova partita e la lista degli utenti invitati. Il client rimane in attesa della conferma dell'invio degli inviti oppure che la partita è stata annullata.

GestionePartita gestisce la partita dal momento in cui si sceglie di visualizzare gli inviti fino a quando si ricevono i risultati della partita. Il thread viene attivato quando viene premuto il bottone VisualizzaInviti ed è composto da due metodi:

- private String setupPartita()
  Se non vi sono inviti da visualizzare mostra un messaggio, altrimenti mostra una finestra in cui è possibile selezionare l'invito. Se tale finestra viene chiusa senza selezionare un invito l'operazione viene annullata, altrimenti viene chiesto se si vuole confermare o rifiutare l'invito. La chiusura della finestra di conferma viene interpretata come il rifiuto dell'invito.
  In entrambi i casi viene aperta una connessione TCP con il server e, dopo l'identificazione dell'utente, viene inviata la conferma/rifiuto dell'invito. In caso di conferma il client invia un messaggio di rifiuto a tutti gli altri inviti prima di eliminarli (in questo modo gli altri client che hanno accettato la partita non rimangano inutilmente in attesa fino alla fine del setup del server).
  Il thread rimane in attesa delle lettere della partita o del messaggio di annullamento partita prima di proseguire restituendo rispettivamente le lettere e null.

- private void avvioPartita(String lettere)
  Se setupPartita ha restituito le lettere, viene lanciato questo metodo, il quale lancia il thread multicast (TwistClientMulticat) che si registra al multicast del server e rimane in attesa dei risultati; una volta ricevuti aprirà un messaggio che mostra i risultati.
  Quando la partita è pronta viene visualizzato un messaggio. Con la chiusura della finestra viene avviato un thread che rimane attivo per due minuti e che apre la finestra di dialogo che mostra le lettere e permette l'input delle parole. Al termine dei due minuti viene chiusa la finestra e viene aperta una connessione UDP con il server tramite la quale viene inviato un messaggio contenente username e l'elenco delle parole separate da "_". A questo punto il client attende che termini il thread del multicast per terminare.
  Dal momento in cui viene confermato un invito fino a quando non si ricevano i risultati o la partita viene annullata, vengono disabilitati i bottoni di nuova partita e visualizza inviti in quanto il client gestisce una sola partita alla volta.
  In caso il client non invii le parole al server entro il tempo stabilito, viene visualizzata la classifica della partita, ma non viene chiusa la finestra di input.
  La chiusura della finestra di input prima dello scadere dei due minuti comporta l'invio al server delle parole inserite e l'attesa dei risulati.

VisualizzaClassifica apre una connessione TCP con il server ed invia username e richiesta prima di attendere la risposta del server. Una volta ricevuta il client mostra una finestra con la classifica.

Attivazione dei thread

Server

TwistServerMain avvia il server RMI e il thread implementato in TwiatServerTCPaccept.

TwiatServerTCPaccept attiva un thread per ogni utente che effettua una richiesta sulla porta PORTA_INIT, implementato dalla classe TwistServerHandler.

Il metodo setupPartita di TwistServerHandler attiva un thread, chiuso dopo 7 minuti per il setup della partita (conferma/rifiuto invito). Il thread viene interrotto chiudendo la server socket in caso tutti i partecipanti abbiano confermato l'invito, oppure un utente invitato ha rifiutato la partita. La receive rimane in attesa dei messaggi per 7 minuti, al termine dei quali viene interrotta.
Questo thread viene implementato dalla classe SetupPartitaHandler.
Il metodo avviaPartita di TwistServerHandler attiva il thread per il server multicasting, implementato nella classe TwistServerMulticast. I due thread comunicano tramite una blockingqueue. Il metodo ricevoRisultati attende la terminazione del thread multicast.

UtentiRegistrati viene creato come una map sincronizzata in quanto viene usato da diversi thread. I metodi della classe Utente sono sincronizzati per la stessa ragione.

Client

TwistClientMain estende la classe TwistClientGui che contiene l'implementazione dell'intefaccia. La classe stessa implementa il thread chiamato dal main con SwingUtilities.invokeLater.
Il metodo printMessage(String message) si TwistClientGUI chiama SwingUtilities.invokeLater(new ShowMessage(message, msgTextArea)) per la scrittura di in messaggio nell'apposita area sull'interfaccia grafica.

GestioneClientGUI gestisce gli action listener. Quando viene premuto il bottone visualizzaClassifica, viene attivato il thread implementato da VisualizzaClassifica, il bottone NuovaPartita avvia il thread NuovaPartita e il bottone visuzlizzaInviti avvia il thread GestionePartita.
Il metodo avvioPartita di GestionePartita avvia il thread implementato da Game il quale apre la fiestra del gioco. Questo thread viene interrotto dopo 2 minuti, con la conseguente chiusura della finestra.
Questo metodo avvia anche il thread implementato da TwistClientMulticast che contiene il client multicast per ricevere i risultati della partita, e attende la sua terminazione prima di terminare il thread GestionePartita.

InvitiRivevuti è la lista degli inviti notificati tramite callback RMI e per questo è condivisa sia dal TwistClientRMI che da Gestione partita. Per renderla thread safe ho usato un Vector.