01/04/2025

**RELAZIONE DI SISTEMI E RETI**

**Progetto autovelox**

**AUTORI: STUDENTI DELLA CLASSE – 4IB**

* Andrea Marturano Sferra

|  |
| --- |
| **ELENCO COMANDI UTILIZZATI** |
| **MOSTRA\_TUTTO**: questo comando, digitato sul terminale, permette di mostrare tutte le informazioni presenti nel file CSV.  **RIGA:** seguito da un valore numerico compreso tra 1 e 1556, se digitato stampa all’utente la riga corrispondente al valore numerico scelto.  **COLONNA:** seguito da un valore numerico compreso tra 1 e 9, se digitato stampa all’utente la colonna corrispondente al valore numerico digitato.  **CERCA\_ID:** seguito da un valore numerico composto da 9 cifre (es: 474673274), permette all’utente di svolgere una ricerca specifica basata sull’identificatore dell’autovelox corrispondente.  **QUIT:** una volta digitato termina inequivocabilmente la connessione tra client e server. |

|  |
| --- |
| **ELENCO CLASSI UTILIZZATE** |
| Classe **Server**: rappresenta il componente principale lato server dell'applicazione client-server. La sua funzione è accettare connessioni dai client, leggere i dati da un file CSV contenente informazioni sugli autovelox e creare un thread dedicato per gestire ciascun client collegato.  Classe **Autovelox**: rappresenta un modello dati (o "entità") per gestire le informazioni relative a un singolo autovelox, lette da un file CSV. Serve a strutturare e incapsulare i dati di ogni rilevatore di velocità.  Classe **GestoreCliente**: ha il compito di gestire la comunicazione tra il server e un singolo client in un'applicazione client-server basata su socket TCP. Estende la classe Thread, il che permette di gestire più client contemporaneamente, assegnando un thread a ciascuna connessione.  Classe **Cliente**: rappresenta il client TCP che si connette al server (gestito dalla classe GestoreCliente) per inviare comandi e ricevere risposte. È la controparte del server in un'applicazione client-server.  Classe **LettoreCSV**: ha il compito di leggere un file CSV contenente i dati degli autovelox e restituire una lista di oggetti Autovelox, ognuno dei quali rappresenta una riga del file (esclusa l’intestazione). |

|  |
| --- |
| **ELENCO METODI PRINCIPALI** |
| **toString():** grazie a questo metodo sono riuscito a stampare le informazioni sul terminale in modo ordinato e chiaro;  **cercaPerId():** questo metodo mi ha permesso di aggiungere la funzionalità della ricerca per identificatore rendendo l’esperienza dell’utente più completa;  **run():** questo metodo si occupa di ricevere comandi da un client collegato via socket, interpretarli, e rispondere con le informazioni richieste, lavorando sui dati contenuti nella lista;  **leggiFile():** questo metodo ha svolto il compito di estrapolare le informazioni dal file CSV e di implementarle nella lista di oggetti;  **startsWith():** mi ha permesso di verificare se una stringa inizia con una determinata sequenza di caratteri. |

**GESTIONE DEGLI ERRORI DI TRASMISSIONE**

**"Errore nel server":** questo messaggio viene stampato se l’utente ha commesso errori durante lo sviluppo della connessione client/server;

**“Indice riga fuori dai limiti":** questo messaggio viene stampato se l’utente ha inserito un valore numerico che non corrisponde alle righe presenti nel file;

**“Formato comando RIGA non valido":** questo messaggio viene stampato se l’utente ha utilizzato un formato NON numerico per identificare una riga;

**"Indice colonna fuori dai limiti":** questo messaggio viene stampato se l’utente ha inserito un valore numerico che non corrisponde alle colonne presenti nel file;

**“Formato comando COLONNA non valido":** questo messaggio viene stampato se l’utente ha utilizzato un formato NON numerico per identificare una colonna;

**“Formato comando CERCA\_ID non valido":** questo messaggio viene stampato se l’utente ha utilizzato un formato scorretto per richiamare un identificatore;

**"Comando non riconosciuto":** questo messaggio viene stampato se l’utente ha digitato nel terminale un comando non esistente;

**"Errore client":** questo messaggio viene stampato se non è stata iniziato correttamente un collegamento dal client al server;

**“Autovelox con ID " + idRicerca + " non trovato":** questo messaggio viene stampato se l’utente sta cercando un autovelox non esistente tramite l’identificatore.

**RICHIESTE NELLA CONSEGNA**

* Leggere e interpretare un file CSV, organizzando i dati in una struttura interna.
* Offrire un’interfaccia di comunicazione remota tramite socket per ricevere richieste e inviare risposte al client.
* Gestire in modo robusto eventuali errori, come richieste non valide o malformate.
* Permettere all’utente di inviare richieste specifiche al server.
* Mostrare in modo chiaro e leggibile i dati ricevuti dal server

**REUISITI TECNICI**

* Utilizzare i socket del linguaggio Java per la comunicazione client/server.
* Garantire che il codice sia ben documentato e conforme alle buone pratiche di programmazione.
* Gestire contemporaneamente più client con l’uso di thread sul lato server.

**ESTENSIONI**

* Implementare funzioni avanzate come la possibilità di filtrare o ordinare i dati.
* Realizzare un client avanzato dotato di interfaccia grafica.
* Implementare analoga applicazione client/server su protocollo UDP.

**DESCRIZIONE DEL CODICE IMPIEGATO**

Il programma Java si compone di due moduli principali: uno per l’elaborazione di file CSV e uno per la gestione della comunicazione client/server basata su protocollo TCP. In fase di avvio, il server apre e legge un file CSV utilizzando le classi standard di input/output di Java, come BufferedReader e FileReader. Ogni riga del file viene suddivisa in campi tramite l’uso della funzione split() applicata al delimitatore, solitamente una virgola. I dati estratti da ciascuna riga vengono poi utilizzati per istanziare oggetti di una classe modellata in modo da rappresentare ciascun record del CSV, i quali vengono memorizzati all’interno di una struttura dati dinamica come ArrayList.

Una volta popolata la lista di oggetti, il server avvia un socket di ascolto tramite la classe ServerSocket, specificando una porta TCP su cui rimanere in attesa di connessioni in ingresso. Per ogni client che si connette, il server crea un nuovo thread utilizzando un Runnable o un’estensione della classe Thread, garantendo così la gestione concorrente di più client. All’interno di ciascun thread, viene mantenuto un canale di comunicazione bidirezionale attraverso l’uso delle classi Socket, InputStreamReader, BufferedReader e PrintWriter, permettendo al client di inviare richieste e ricevere risposte in tempo reale. Quando un client invia una richiesta di visualizzazione dei dati, il server formatta la lista degli oggetti in stringhe leggibili e le invia attraverso il socket. L’interazione continua fino a quando il client non chiude la connessione, momento in cui il thread dedicato termina la propria esecuzione. Questo modello garantisce un accesso simultaneo e sicuro da parte di più client alla stessa base dati estratta dal file CSV.