### **Relazione del Progetto: Applicazione Client-Server per la Consultazione Remota di un File CSV**

#### **Obiettivo del Progetto**

Il progetto consiste nella realizzazione di un'applicazione client-server in linguaggio Java che consenta di gestire un file CSV contenente informazioni relative a strutture ricettive alberghiere. L'applicazione è progettata per consentire la consultazione remota dei dati attraverso una comunicazione via socket. Il server gestisce la lettura di un file CSV, lo carica in memoria e risponde alle richieste dei client, che possono interrogare i dati o visualizzarli. La comunicazione tra client e server avviene tramite il protocollo TCP, con comandi ben definiti.

#### **Struttura dell'Applicazione**

L'applicazione è composta da tre classi principali:

* **Server** : gestisce la connessione con i client e carica i dati da un file CSV.
* **GestoreClient** : gestisce le richieste dei singoli client, elaborando i comandi e rispondendo.
* **Client** : interagisce con l'utente, inviando comandi al server e visualizzando i dati ricevuti.

### **Elenco Comandi Disponibili**

Il client può inviare i seguenti comandi al server:

* **GET\_ROW** numero a scelta: recupera la riga corrispondente al numero numero.
* **GET\_COLUMN** numero a scelta: recupera i valori della colonna corrispondente al numero numero.
* **SHOW\_ALL**: visualizza tutte le righe del CSV.
* **exit**: chiude la connessione con il server.

**Esempi del comando GET\_…**

* **GET\_ROW** 5: recupera la riga con indice 5.
* **GET\_COLUMN** 2: recupera tutti i valori della colonna 2.

### **Classi dell'Applicazione**

#### **Classe Server**

Questa classe gestisce il server che riceve le richieste dai client e invia le risposte.

**Metodi principali:**

* **main**: Inizializza il server e ascolta le connessioni sulla porta predefinita (12345). Per ogni client in arrivo, viene avviato un nuovo thread GestoreClient.
* **caricaCSV**: Carica i dati dal file CSV nella lista datiStrutture per poterli consultare e manipolare.

**Spiegazione:**

* Il server si avvia sulla porta 12345(nome scelto per il mio server).
* Quando un client si connette, viene creato un thread GestoreClient che gestisce la comunicazione.
* Il file CSV viene letto riga per riga e ogni riga viene suddivisa nei singoli campi tramite il delimitatore ;, aggiungendo i dati alla lista datiStrutture.

#### **Classe GestoreClient**

Questa classe è un gestore per ogni client, implementata come un thread separato che comunica con il client e interpreta i comandi.

**Metodi principali:**

* **run**: Gestisce il ciclo di vita della connessione con il client, ricevendo i comandi e inviando le risposte.
* **gestisciComando**: Interpreta i comandi inviati dal client e restituisce i risultati.  
  + GET\_ROW: Restituisce una specifica riga del CSV.
  + GET\_COLUMN: Restituisce i dati di una specifica colonna.
  + SHOW\_ALL: Mostra tutte le righe del file CSV.

**Gestione degli Errori:**

* Viene gestita l'eccezione NumberFormatException per i comandi che richiedono un numero (ad esempio, GET\_ROW <n>).
* Se un comando è errato o il numero di riga o colonna è fuori dai limiti, viene inviato un messaggio di errore.

#### **Classe Client**

Questa classe simula il comportamento di un client che si connette al server, invia comandi e visualizza le risposte.

**Metodi principali:**

* **main**: Gestisce l'interazione con l'utente, inviando i comandi e ricevendo le risposte dal server.
* **Input da tastiera**: Permette all'utente di inserire comandi tramite la console.

**Spiegazione:**

* Dopo essersi connesso al server, il client stampa un messaggio di benvenuto.
* L'utente può inserire uno dei comandi specificati.
* Il client invia il comando al server e visualizza la risposta ricevuta.
* Se il comando è exit, il client termina la connessione.

**Gestione degli Errori:**

In caso di errore di connessione al server o durante l'invio dei comandi, vengono catturati e stampati messaggi di errore.

### 

### 

### 

### **Gestione degli Errori nell'Applicazione**

* **File CSV**: Se il file CSV non viene trovato o ci sono problemi durante la lettura, viene stampato un messaggio di errore.
* **Comandi**: Se un comando è malformato o richiede una riga/colonna che non esiste, il server invia un messaggio di errore.
* **Connessione**: Se si verifica un errore durante la connessione, viene stampato un errore sul lato client o server, ma l'applicazione continua a funzionare.

### **Protocollo di Comunicazione**

Il protocollo di comunicazione tra client e server è semplice e ben definito:

1. **Comando dal Client**: Il client invia una stringa di comando tramite il socket.
2. **Elaborazione del Comando**: Il server riceve il comando, lo elabora e invia la risposta tramite il socket.
3. **Risposta al Client**: La risposta del server può essere:

* I dati richiesti (una riga o una colonna).
* Un messaggio di errore, nel caso di comando malformato o indici non validi.

### **Considerazioni sulla Scelta del Protocollo**

Abbiamo scelto di utilizzare TCP per la comunicazione client-server, in quanto:

* **Affidabilità**: TCP garantisce che i dati vengano ricevuti in modo ordinato e senza perdita, fondamentale per il corretto funzionamento della comunicazione tra client e server.
* **Semplicità di implementazione**: La gestione dei socket TCP in Java è semplice e permette di concentrarsi sulla logica di elaborazione dei comandi.

### 

### **Possibili Estensioni**

* **Filtraggio e Ordinamento dei Dati**: Si potrebbe implementare la possibilità di ordinare o filtrare i dati (ad esempio, per località o tipo di struttura).
* **Interfaccia Grafica del Client**: Il client potrebbe essere dotato di un'interfaccia grafica per migliorare l'esperienza utente, con comandi predefiniti e visualizzazione dei dati in una tabella.

### **Conclusioni**

Il progetto ha consentito di sviluppare un'applicazione client-server che gestisce e permette la consultazione remota di un file CSV contenente informazioni su strutture ricettive. La struttura del codice è semplice e modulare, con una gestione robusta degli errori. Sono state anche esplorate possibili estensioni, come l'implementazione di un'interfaccia grafica e l'uso di UDP per la comunicazione.