Relazione HOTELIER

Progetto di Laboratorio III

Giovanni Giuseppe Pinna

Programmi e scelte implementative

Elenco dei programmi utilizzate nel progetto:

Main Server

Il programma che avvia il server, legge i parametri salvati nel *server.properties* ossia il configFile con i dati utili al server, inizializza le strutture dati per gli utenti (**Umap**) e per gli hotel (**Hmap**) e con le analoghe funzioni **user_reader()** e **hotel_reader()** legge i dati salvati nei rispettivi json (*user.json* e *Hotels.json*) ed infine crea un thread che gestisce la terminazione con **TerminationHandler** in caso di interruzione da terminale.

- **readConfing()** : prende il file *server.properties* e legge ogni campo all'interno del file, con le funzioni `prop.getProperty(String key)`.
- **Avvio del server** : Creo una ServerSocket con `new ServerSocket(port);` che si mette in ascolto alla porta numero 8053 (intero salvato in *port* nel confing file del server), e col comando `socket_list.setReuseAddress(true);` permetto alla socket di riutilizzare un indirizzo locale anche se ancora in uso.
- **Connessione** : Quando un client si collega alla porta, il server con `socket_list.accept()` accetta la connessione e il **MainServer** gestisce questa connessione con un task (**ServerTask**) gestito da un thread della *CachedThreadPool* pool grazie al comando `pool.execute()` .
- **Gestione interruzioni** : Per chiudere il server viene lanciato dal terminale il comando *ctrl+C*, grazie al `Runtime.getRuntime().addShutdownHook()` viene fatto partire un thread che gestisce **TerminationHandler** per chiudere il server e salvare i dati nei appositi file json.
- **user_Reader()** : Questa funzione prende il file *users.json* dove sono salvati i dati di tutti gli utenti registrati su Hotelier.

Viene creato un json reader con `JsonReader reader = new JsonReader(new FileReader(upath))`, si controlla che il file non sia vuoto e si apre la apre l'array con `reader.beginArray()`.

Per ogni oggetto viene letto rispettivamente: nome, password, punti esperienza e badge. Grazie al comando `reader.nextName();` si prende il nome del campo e con `reader.nextInt()` o `reader.nextString()` i valori .

Infine viene creato l'utente `u = new User(username, password);`, ripristinati i punti esperienza ed il badge ed infine inseririto nella **Umap**, per l'inserimento si usa il comando `put(String username, User u)` non atomico perchè il MainServer è l'unico thread che esegue il ripristino della hash e perciò non vi è nessuna race condition.

- **hotel_Reader()** : Analogo per gli hotel, vengono letti per ogni oggetto dell'array: i vari dati dell'hotel, però nella parte delle recensioni, il file iniziale contiene un oggetto contente i voti settati a 0, per prefenza personale ho preferito creare un array di oggetti vuoto, invece se il file contiene recensioni viene letto come come di consueto. Nemmeno il ranking è stato salvato nel file iniziale e in quel caso viene settato a 0. Viene creato l'hotel, inserito nella lista della sua citta e poi la lista viene aggiornata nella Hmap.

Prima di essere inserito nella Hash.

Dopo essere stato caricato nella Hash e poi salvato nel json.

Server Task

Questo task gestisce la connesione tra client e server, lo scambio di messaggi avviene grazie ad una connesione TCP con i i flussi *InputStream* e *OutputStream*.

- **run()** : il task si tratta di ricevere un intero dal client nel flusso di input *in* col comando `int scelta = in.readInt();` e in base a quella eseguire le funzioni scelte, questo avviene grazie ad uno `switch(scelta)`. Nel caso 8, ossia il client decide di

uscire dall'applicazione viene chiusa la socket con `socket.close();` e terminato il task.

- **register()** : funzione dove il client invia uno username `String username = in.readUTF();`, il server controlla che lo username non sia già salvato nella **Umap** grazie alla funzione `containsKey(String username)` , per ogni casistica il server invia un apposito messaggio al client.

  Se lo username non è presente si può ricevere la password e se quest'ultima non è vuota, viene creato lo username e salvato nella map.

  Da evidenziare il fatto che a differenza di **user_Reader()** viene utilizzato il comando `putIfAbsent(String username, User u)` che permette un inserimento atomico nella **Umap**, per evitare che di avere problemi di sincronizzazione.
- **login()** : molto simile al **register()**, il client invia uno username, si controlla che sia presente nella **Umap**, e l'utente deve inviare una password corretta entro 3 tentativi. Sia register che login non possono essere effettuate se l'utente è gia loggato.
- **searchHotel()** : il server riceve una stringa dal client che indica la citta, si controlla che esista la nella **Hmap**.

```
// controllo se la città è presente nella hashmap
ArrayList<Hotel> list = Hmap.get(city);
if (list == null) {
    out.writeUTF(str:"CNE"); //CNE = City Not Exist
    return;
}
out.writeUTF(str:"CE"); //CE = City Exist
```

  In caso affermativo viene inviata dal client la stringa contenente il nome completo dell'hotel, si scorre la lista associata alla città e se esiste l'hotel, si creano una stringa contente tutte le informazioni salvate nella hash e poi inviata al client.
- **searchAllHotel()** : Praticamente uguale al precedente solo che prende la citta, e invia tutti gli hotel della lista associata.
- **insertReview()** : Viene preso un hotel con le stesse operazioni precedentemente elencate, e vengono presi i voti inviati dal client.

```
// Aggiungo la recensione alla lista di recensioni dell'hotel
g = new Giudizi(pulizia, posizione, servizi, qualità, user, data);
h.set_voti(g);
```

Ho creato l'oggetto **Giudizi** e inserito nella lista delle recensione assieme allo username di chi lo ha inserito e la data. Inoltre vengono incrementati i punti di esperienza dell'utente e modificato il rank e la lista in base al ranking con la funzione **list_sort()** inviando la data della recensione.

- **showBadges()** : viene inviato al client il badge dell'utente associato, a patto che l'utente sia loggato.
- **list_sort()** : funzione che viene invocata da insertReview() dopo che ha modificato i ranking.

  Viene creato un comparatore `Comparator<Hotel> comparator` che con la funzione `compare(Hotel h1, Hotel h2)` prende due hotel e li confronta in base al ranking, in caso di ranking uguale la funzione ordina in base alla data di inserimento più recente.

```
Comparator<Hotel> comparator = new Comparator<Hotel>() {
  @Override
  public int compare(Hotel h1, Hotel h2){
    int rank_compare = Double.compare(h1.get_ranking(),h2.get_ranking())
    if(rank_compare != 0){
      return rank_compare;
    }
    // Se i voti sono uguali, ordina per data di inserimento
    return h1.get_data().compareTo(h2.get_data());
  }
};
```

Con questa funzione di compare viene eseguito il *sort* della lista associata alla citta e viene controllato se l'ultimo valore della lista è lo stesso della lista prima dell'ordinamento. Ho preferito inserire i migliori hotel in fondo alla lista per il semplice motivo che nel terminale poichè si scorre dal basso verso l'alto, era più comodo vedere subito gli hotel migliori dal basso.

```java
if(!(new_max.equals(old_max))){
  // Se l'hotel migliore è cambiato invio un messaggio di notifica

  // Creo un datagramsocket con il try with resources
  try(DatagramSocket socket = new DatagramSocket(port)){
    //Stringa contenente il messaggio da inviare
    String news = "Attenzione, l'"+new_max+" è diventato il miglior hotel di "+citta+"!";
    InetAddress group = InetAddress.getByName(mcadd);
    // Creo il pacchetto da inviare
    DatagramPacket response = new DatagramPacket(news.getBytes(), news.getBytes().length, group,mcport);
    // Invio il pacchetto
    socket.send(response);
  } catch (Exception e){
    e.printStackTrace();
  }
```

Nel caso di hotel migliore viene creato un *DatagramSocket*, ossia viene creata una connesione *UDP* che permette a tutti i client loggati di ricevere la notizia che in una determinata città un nuovo hotel è primo nella classifica. La creazione del *DatagramSocket* avviene con un *try with resources* con il numero di porta (`port`) utilizzato per la connesione tra client e server, viene ottenuto l'indirizzo ip del gruppo multicast con `InetAddress.getByName(mcadd);` con `mcadd` l'indirizzo multicast salvato nel confingFile. Incapsulo la stringa nel DatagramPacket e lo invio con `socket.send(DatagramPacket response)`. Infine aggiorno la lista con `Hmap.replace(String città, ArrayList<Hotel> newlist)`.

Termination Handler

Task invocato dal MainServer quando riceve un interruzione, chiude il server, termina il threadpool con `shutdown()` e salva i dati nelle hash.

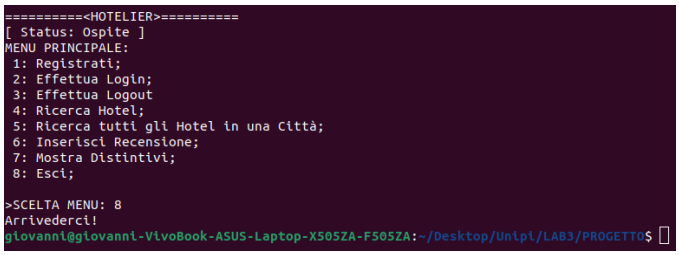
- **user_Writer()**: crea una json writer `JsonWriter writer = new JsonWriter(new FileWriter(upath));` che inserisce i dati dell'utente già citati in User_reader.
- **hotel_writer()**: analogo per hotel, vengono salvati i dati dell'hotel contenuti nella hash. Da evidenziare che la data, viene salvata come stringa poichè non è possibile salvare nel json valori di tipo Date.

```java
Date d = g.get_data();
SimpleDateFormat format = new SimpleDateFormat(pattern:"yyyy-MM-dd HH:mm:ss");
String format_d = format.format(d);
writer.name(name:"data").value(format_d);
writer.endObject();
```

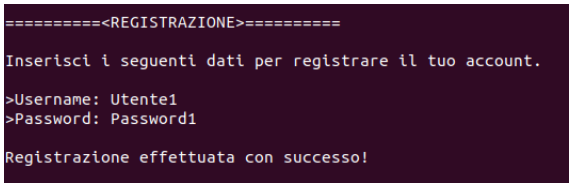
Main Client

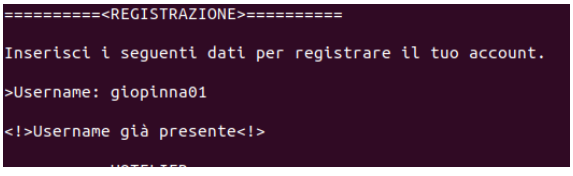
Programma client che permette ai vari utenti di utilizzare Hotelier. Appena aperto il programma si connette al server tramitre socket al numero `port` e indirizzo `address` salvato nel confingFile *client.properties*. Viene aperto un menù con tutte le funzionalità a cui l'utente può accedere, per gli aggiornamenti sui nuovi hotel primi in classifica viene creato un thread incaricato di attendere il segnale dal server.

- **main()** : prende da terminale l'intero che indica quale funziolità l'utente vuole utilizzare. Nel caso 8, l'utente ha finito di utilizzare Hotelier. Viene chiusa la socket e interroto il thread suglia aggiornamenti.



- **readConfig**: uguale all'omonima funzione nel **MainServer** per tutti i parametri salvati nel *client.properties*.
- **register()**: In questa funzione l'utente inserisce lo username, che viene inviato al server tramite il flusso *DataOutStream* con la funzione `out.writeUTF(String username)`. Nel caso il server risponde con l'username non esiste, allora può essere inserita anche la password. Questa funzione può avvenire solo se non si è ancora loggati.





- **login()**: L'utente inserisce username e password, il server controlla che lo username esista e la password combaci, in caso affermativo viene settato a true un valore booleano chiamato `log` che indica che l'utente si sia loggato.

```java
s = in.readUTF();
if (s.equals(anObject:"CPA")) { //CPA = Correct PAssword
  // Se la password è corretta, il client viene loggato
  user = username;
  System.out.println("\nBentornato " + user);
  log = true;
  // Creazione del newsThread
  NewsTask newsH = new NewsTask(mcadd, mcport, timer);
  newsThread = new Thread(newsH);

  // Avvio del newsThread
  newsThread.start();
```

Inoltre viene fatto partire un thread col task **NewsTask**, rimane in attesa del segnale di nuovo hotel in prima posizione.

Anche qui vi è presente un `Runtime.getRuntime().addShutdownHook()` per permettere di interrompere il thread che esegue **NewsTask**, in caso di interruzione del **MainClient**.

```
==========LOGIN==========
Inserisci i seguenti dati per poter accedere!

>Username: Utente1
>Password: Password
<!>Password errata, tentativi rimanenti: 3<!>
>Password: Password1

Bentornato Utente1
```

```
==========LOGIN==========
Inserisci i seguenti dati per poter accedere!

>Username: Utente2

<!>Username non presente<!>
```

- **logout()** : L'utente si disconnette e la variabile log torna a false, interrompo il thread **NewsTask** perchè non sono più loggato.
- **searchHotel()**: L'utente inserisce citta e nome dell'hotel, se l'hotel viene trovato, il server la stringa con le informazione dell'hotel e stampato a terminale.

```
==========RICERCA HOTEL==========
Inserisci citta e nome dell'hotel che desideri vedere.

>Città: Cagliari
>Hotel: Hotel Cagliari 4

------------------------------
Hotel trovato: Hotel Cagliari 4
Descrizione: Un organizzato hotel a Cagliari, in Via Italia, 72
Città: Cagliari
Telefono: 344-8099110
Servizi: [TV in camera, Sauna]
Voto: 2.5
Giudizi: [
{       Pulizia: 3;     Posizione: 4;   Servizi: 2;     Qualità: 1}
Inserito da: gio in data: [Tue May 14 14:31:56 CEST 2024]

]

------------------------------
```

- **searchAllHotel()**: Analogo ma con tutti gli Hotel.

- **insertReview()** : come i precedenti, inserisco città e nome dell'hotel, in caso affermativo inserisco le recensioni per l'hotel selezionato, invio i dati al server che a sua volta risponde con il calcolo del punteggio complessivo aggiornato ed i punti esperienza ottenuti con la recensione. Non è possibile inserire recensione da ospite.

```
==========INSERIMENTO RECENSIONE==========

>Città: Cagliari
>Nome: Hotel Cagliari 5
>Pulizia: 5
>Posizione: 2
>Servizi: 3
>Qualità: 4
Media recensione: 3.0
Voto Complessivo Hotel Aggiornato: 3.5
Recensione inserita con successo!

Sono stati incrementati i tuoi punti di 50 punti
 Punti exp attuali: 7050
```

- **showBadges()**: la funzione invia lo username dell'utente, e il server risponde con il badge associato. Non è possibile effettuare questa operazione da ospite.

News Task

Questo task gestisce l'attesa in ascolto del segnale del nuovo hotel in prima posizione nella classifica di una città.

```
try {
  ms = new MulticastSocket(mcport);
  ms.setSoTimeout(timer);
  group = InetAddress.getByName(address);
  ms.joinGroup(group);
  while (!Thread.currentThread().isInterrupted()) {
    DatagramPacket dp = new DatagramPacket(new byte[1024], length:1024);
    try{
    ms.receive(dp);
    news = new String(dp.getData(), offset:0, dp.getLength());
    if(!news.equals(anObject:"")){
      synchronized(System.out){
      System.out.println("\n=News: " + news+"=\n");
      }
      news = "";
    }
    }catch(SocketTimeoutException e){
     // System.out.println("Timeout scaduto");
    }
  }
```

Viene creato un socket multicast sulla porta specificata (`mcport`). Questo socket verrà utilizzato per ricevere i pacchetti multicast inviati nel gruppo. Viene impostato un timeout per il socket in modo che la ricezione di un pacchetto non blocchi indefinitamente il thread. Se non viene ricevuto alcun pacchetto entro il tempo specificato (`timer`), viene sollevata un'eccezione `SocketTimeoutException`.

Il socket si unisce al gruppo multicast specificato utilizzando `ms.joinGroup(group)` con `group` l'indirizzo IP del gruppo ottenuto da `InetAddress.getByName(address)`.

Eseguo un ciclo con `!Thread.currentThread().isInterrupted()` che esegue il ciclo sino a quando non viene interroto il thread. Viene creato un nuovo *DatagramPacket* per contenere i pacchetti ricevuti.

Con `ms.receive(DatagramPacket dp)` viene ricevuto il messaggio e convertito in stringa per poi essere stampato. Sia nel **NewsTask** che nel **MainClient** viene effettuata una sincrozzazione nel *system.out* per evitare sovrapposizioni nella stampa su terminale. Quando il task viene interroto, il ciclo termina e il socket esce dal gruppo e infine chiuso.

User.java

Classe contenente gli attributi degli utenti ed i rispettivi metodi get e set.

Hotel.java

Classe contentente gli attrubuti degli hotel ed i rispettivi metodi get e set.

- **set_rank()** : funzione che riceve la data dell'ultimo voto inserito e calcola il rank, inanzitutto vengono creati dei pesi w (weight) per i voto (*50%*), la data (*30%*) e il numero di recensioni (*20%*) e poi viene fatta la somma pesata per dei vari attributi moltiplicati con il loro rispettivo peso.

$$Rank = \text{media voto} \cdot 0,5 + \text{differenza tra date} \cdot 0,3 + \text{num recensioni} \cdot 0,2$$

Giudizi.java

Classe contentente i voti delle recensioni ed i rispettivi metodi get.

Confing File

client.properties

```
#
# File di configurazione per i
#

# Numero porta
port = 8053
```

server.properties

```
#
# File di configurazione per i
#

# Porta del server
port = 8053
```

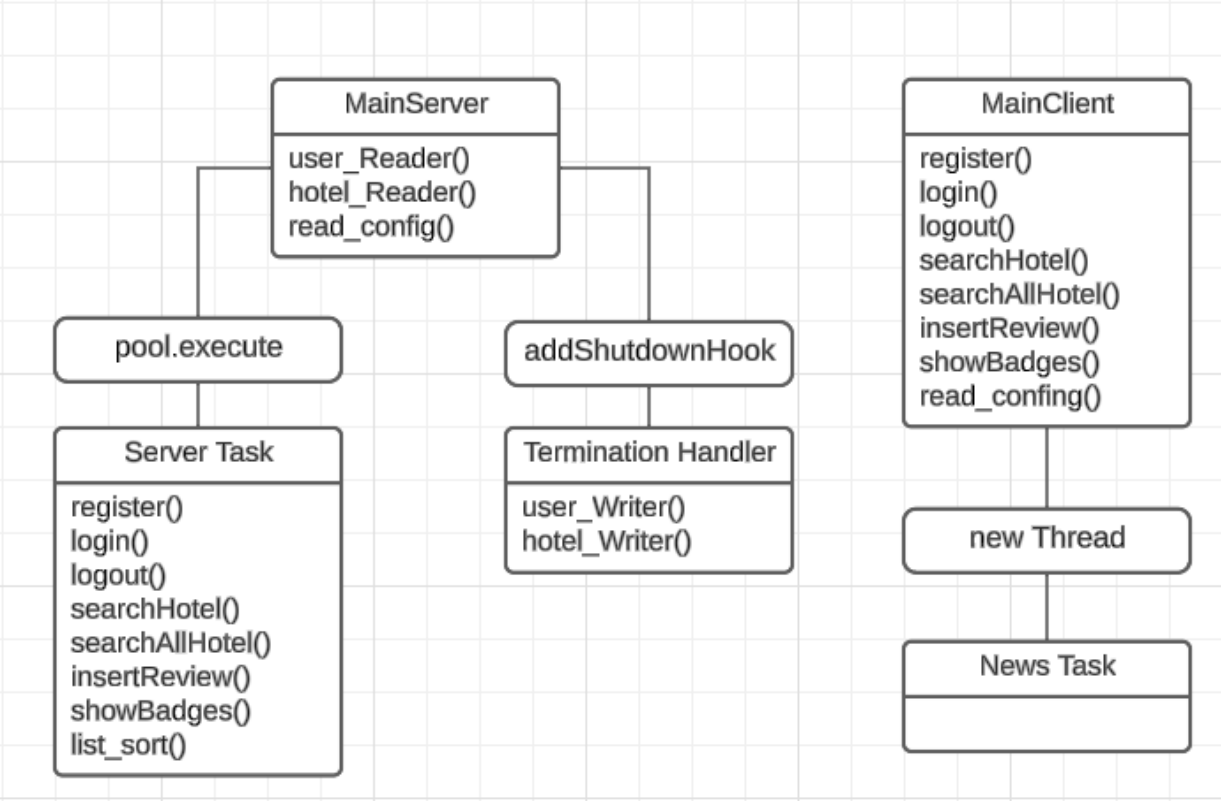
```
# Numero porta multicast
multicastPort = 6180

# timer socket multicast
timer = 1000

# Indirizzo IP del server
address = localhost

# Indirizzo multicast
multicastAddress = 224.0.0.1
```

```
# Numero porta multicast
multicastPort = 6180

# Indirizzo IP del server
address = localhost

# Indirizzo multicast
multicastAddress = 224.0.0.1

# Path file json utenti
upath = users.json

# Path file json hotel
hpath = Hotels.json
```

Strutture dati utilizzate

- **ConcurrentHashMap**
  - **ConcurrentHashMap<String, User> Umap**: Contiene gli utenti associati al loro username.
  - **ConcurrentHashMap<String, ArrayList<Hotel>> Hmap** : Contiene le liste degli hotel che stanno in una determinata città.
  - **ArrayList<Hotel> list** : lista di hotel.
  - **ArrayList<Giudizi> glis** : lista di giudizi.

Schema thread



Approcci di sincronizzazione

Nelle **ConcurrentHashMap Hmap** e **Umap** devono essere utilizzati funzioni apposite per evitare che si possano effettuare scritture contemporaneamente.

- **Scrittura**: per scrivere nella Umap uso la funzione `putIfAbsent(String username, User u)` che inserisce uno username e l'utente associato nella hash a patto che lo username (chiave della hash) inserito non sia già stato precedentemente inserito. Nella **Hmap** invece non devo inserire nuovi hotel ma devo modificare le liste dopo aver calcolato il *ranking* in base alle nuove recensione, viene utilizzata la funzione `replace(String città, ArrayList<Hotel> list)` che rimpiazza la nuova lista aggiornata, la città deve già essere presente nella hash.
- **Lettura**: la lettura può essere effettuata da molteplici thread contemporeaneamente, può essere tranquillamente usata la funzione `get(String username)`.
- **Stampe**: con l'utilizzo del thread **News Task** ho notato che quando viene ricevuto il messaggio di nuovo miglior hotel, vi era una sovrapposizione nelle stampe sul terminale. Perciò ho deciso di implementare un monitor per garantire

che un thread alla volta possa stampare da terminale. Col comando `synchronized(System.out)` utilizzo System.out come monitor.

Salvataggio dei Dati

Come descritto nel **Termination Handler** non effettuo un salvataggio pediodico dei dati dalle hash ai file json, ma bensì lo effettuato solo nel caso di terminazione del server.

Istruzioni per l'uso

- **Compilazione**: da terminale, nella directory corretta, deve essere eseguito il comando

  ```
  javac -cp lib/gson-2.10.1.jar *.java
  ```

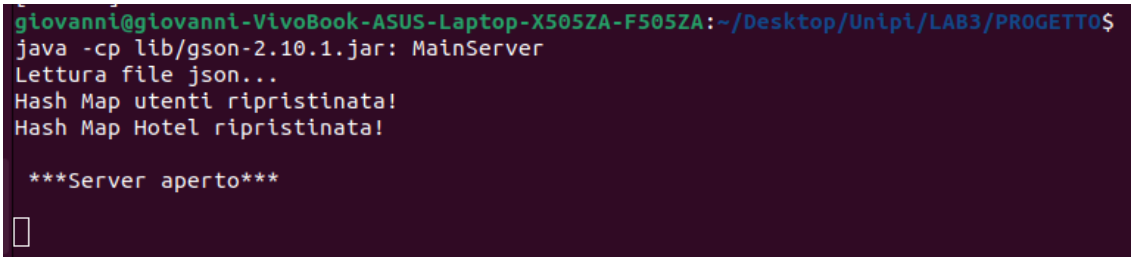
  Questo permette di compilare tutti i file *.java*, il `-cp` indica che per essere compilato si ha bisogno anche delle librerie contenute nel file *.jar*.

- **Esecuzione**: da terminale, nella directory del progetto, deve essere eseguito il comando

  - **Server**

    ```
    java -cp lib/gson-2.10.1.jar: MainServer
    ```

    Non richiede alcun dati in input, all'inizio verrano stampati dei messaggi che indicano la lettura dei file json ed il ripristino delle hashmap.

    Per la chiusura del Server deve essere lanciato il *SIGINT* con *ctrl+c*, il server cattura il segnale e come precedentemente spiegato in **Main Server** viene creato il **Termination Handler** che salva le strutture dati e chiude la socket.

```
^C
 Chiusura in corso...
Dati utenti salvati
Dati hotel salvati
[SERVER] Terminato.
```

- **Client**

```
java -cp lib/gson-2.10.1.jar: MainClient
```

Attenzione, il client deve essere eseguito dopo che viene aperto il server, altrimenti viene restituito un messaggio di errore indicando che non è possibile collegare il client al server.

```
giovanni@giovanni-VivoBook-ASUS-Laptop-X505ZA-F505ZA:~/Desktop/Unipi/LAB3/PROGETTO$ java -cp lib/gson-2.10.1.jar: MainClient
<!> Errore nel collegamento al server <!>
java.net.ConnectException: Connection refused (Connection refused)
        at java.base/java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.base/java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:412)
        at java.base/java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:255)
        at java.base/java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:237)
        at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.base/java.net.Socket.connect(Socket.java:609)
        at java.base/java.net.Socket.connect(Socket.java:558)
        at java.base/java.net.Socket.<init>(Socket.java:454)
        at java.base/java.net.Socket.<init>(Socket.java:231)
        at MainClient.main(MainClient.java:33)
```

Nel caso di server aperto, viene mostrato il menù principale.

Per la chiusura basta inserire il numero 8 che come indicato nel menù indica l'uscita dal programma.

```
giovanni@giovanni-VivoBook-ASUS-Laptop-X505ZA-F505ZA:~/Desktop/Unipi/LAB3/PROGETTO$ java -cp lib/gson-2.10.1.jar: MainClient

==========<HOTELIER>==========
[ Status: Ospite ]
MENU PRINCIPALE:
 1: Registrati;
 2: Effettua Login;
 3: Effettua Logout
 4: Ricerca Hotel;
 5: Ricerca tutti gli Hotel in una Città;
 6: Inserisci Recensione;
 7: Mostra Distintivi;
 8: Esci;

>SCELTA MENU: 8
Arrivederci!
```