# LAB3 RELAZIONE

## Introduzione

### Scopo del progetto

Il progetto "HOTELIER: an HOTEL advisor service" ha come scopo principale la realizzazione di un servizio di recensioni per hotel che si ispira al noto sito TripAdvisor, ma con funzionalità ridotte e semplificate. L'obiettivo è quello di creare una piattaforma che consenta agli utenti di registrarsi, effettuare il login e recensire hotel situati nelle città capoluogo delle 20 regioni italiane. Il progetto mira a fornire una base solida per un sistema di valutazione e classifica degli hotel, che consideri parametri come la qualità, la quantità e l'attualità delle recensioni.

### Obiettivi principali

Gli obiettivi principali del progetto sono:

1. Implementazione di un sistema di registrazione e login:
    - Consentire agli utenti di creare un account e accedere al servizio per poter pubblicare recensioni e visualizzare i propri distintivi.
2. Gestione delle recensioni:
    - Permettere agli utenti di inserire recensioni degli hotel, assegnando punteggi sintetici e specifici per le categorie di Posizione, Pulizia, Servizio e Prezzo.
3. Calcolo e aggiornamento del ranking:
    - Implementare un algoritmo per il calcolo del ranking degli hotel basato sui punteggi delle recensioni, che tenga conto della qualità, quantità e attualità delle stesse.
4. Notifiche e distintivi:
    - Attribuire distintivi agli utenti in base al numero di recensioni effettuate e inviare notifiche agli utenti che

hanno eseguito il login quando il ranking locale degli hotel cambia.

### Descrizione generale del progetto

Il progetto "HOTELIER" prevede la realizzazione di due componenti principali:

1. `HOTELIERCustomerClient`:
    - Un client che gestisce l'interazione con l'utente tramite un'interfaccia a linea di comando (CLI). Le operazioni principali del client includono la registrazione, il login, la ricerca di hotel, l'inserimento di recensioni e la visualizzazione dei distintivi.
2. `HOTELIERServer`:
    - Un server che gestisce la registrazione e il login degli utenti, memorizza le informazioni relative agli hotel e agli utenti, riceve e memorizza le recensioni, aggiorna il ranking degli hotel e invia notifiche agli utenti che hanno eseguito il login. Il server carica inizialmente la descrizione degli hotel da un file JSON e persiste periodicamente le strutture dati utilizzate.

## Descrizione del Progetto

### Architettura generale del sistema

Il sistema HOTELIER è progettato seguendo un'architettura client-server, con il client che gestisce l'interazione utente e il server che gestisce la logica di business e la persistenza dei dati. Il client comunica con il server utilizzando protocolli di rete, permettendo agli utenti di registrarsi, effettuare il login, inserire recensioni e consultare i dati degli hotel.

### Descrizione dei componenti principali

Il progetto è suddiviso in diverse componenti, ciascuna con ruoli specifici:

- Client:

    - Comprende le classi `HOTELIERClientMain`, `ClientHelper` e `InputValidator`, che gestiscono l'interfaccia a linea di comando e la validazione degli input.
- Server:
    - Include le classi `HOTELIERServerMain`, `DataStore`, `HotelManager`, `RankingAlgorithm`, `UserManager` e `WorkerThread`, che si occupano della gestione degli hotel, degli utenti e delle recensioni, nonché del calcolo dei ranking.
- Config:
    - Contiene file JSON per la configurazione del client e del server, e per la memorizzazione dei dati degli hotel e degli utenti.
- H_U_R:
    - Comprende le classi `Hotel`, `User` e `Review`, che definiscono le strutture dati utilizzate per rappresentare gli hotel, gli utenti e le recensioni.
- Libs:
    - Contiene la libreria esterna `gson-2.11.0.jar` utilizzata per la manipolazione dei file JSON.

## Tecnologie utilizzate

Il progetto utilizza diverse tecnologie e librerie per la sua realizzazione:

- Java:
    - Linguaggio di programmazione principale utilizzato per implementare sia il client che il server.
- Gson:
    - Libreria per la manipolazione dei file JSON, essenziale per la persistenza e il recupero dei dati.
- Protocolli di rete TCP/UDP:
    - Utilizzati per la comunicazione tra client e server, con il TCP per le operazioni principali e l'UDP per le notifiche multicast.
- File di configurazione JSON:

- Utilizzati per configurare parametri come numeri di porta, indirizzi e intervalli di aggiornamento dei ranking.

# Analisi del Codice

## Strutture Dati Principali:

ConcurrentHashMap:

- `users`:
  - Mappa che memorizza gli utenti registrati, con chiave come username e valore come oggetto `User`.
- `hotels`:
  - Mappa che memorizza gli hotel per città, con chiave come nome della città e valore come lista di oggetti `Hotel`.

## Panoramica delle varie classi

### Client

- `HOTELIERClientMain`:
  - Punto di ingresso del client, gestisce la connessione e le interazioni dell'utente.
- `InputValidator`:
  - Responsabile della validazione degli input utente.
- `ClientHelper`:
  - Gestisce le comunicazioni con il server.

### Server

- `HOTELIERServerMain`:
  - Ascolta le connessioni TCP dai client, gestisce le richieste dei client tramite thread, carica e memorizza i dati degli utenti e degli hotel da file JSON, e mantiene i dati aggiornati in modo persistente utilizzando una classe `DataStore` che esegue periodicamente il salvataggio dei dati.
- `DataStore`:

    - Questa classe implementa l'interfaccia `Runnable` e gestisce il salvataggio periodico dei dati degli utenti e degli hotel in file JSON. Utilizza due `ConcurrentHashMap` per mantenere le informazioni su utenti e hotel.
- `HotelManager`:
    - Gestisce le operazioni relative agli hotel, come la ricerca di hotel per nome e città, l'inserimento di recensioni e l'invio di notifiche al client via UDP.
- `UserManager`:
    - Gestisce le operazioni relative agli utenti, come la registrazione, il login, il logout, e la gestione dei badge.
- `WorkerThread`:
    - Implementa `Runnable` e gestisce le richieste dei client. Utilizza `UserManager` e `HotelManager` per elaborare le richieste e inviare le risposte appropriate ai client.

## H_U_R

Il codice è composto da tre classi principali: `Hotel`, `Review` e `User`

- Classe `Hotel`:
    - Questa classe rappresenta un hotel e contiene vari attributi come:
        1. ID
        2. nome
        3. descrizione
        4. città
        5. numero di telefono
        6. servizi offerti
        7. punteggio di valutazione
        8. valutazioni specifiche
        9. recensioni degli utenti.
    - Fornisce metodi per ottenere e impostare questi attributi, oltre a metodi per rappresentare l'oggetto in formato stringa e JSON.

- Classe `Review`:
  - Rappresenta una recensione lasciata da un utente. Contiene:
    1. punteggi specifici per vari aspetti dell'hotel (pulizia, servizi, posizione, qualità)
    2. punteggio globale
    3. data della recensione.
  - Include metodi per ottenere e impostare questi attributi e per rappresentare la recensione in formato stringa.
- Classe `User`:
  - Rappresenta un utente che può lasciare recensioni sugli hotel. Contiene attributi come:
    1. nome utente
    2. password
    3. badge
    4. numero di recensioni lasciate
    5. stato di login.
  - Include metodi per ottenere e impostare questi attributi e per rappresentare l'utente in formato stringa.

## Funzionalità principali del codice

### Client

Le funzionalità principali del client sono:

1. Registrazione e login:
   - Gli utenti possono registrarsi e accedere al sistema.
2. Ricerca hotel:
   - Gli utenti possono cercare un hotel specifico o tutti gli hotel in una città.
3. Inserimento recensioni:
   - Gli utenti possono inserire recensioni per un hotel.
4. Visualizzazione badge:
   - Gli utenti possono visualizzare i propri badge.
5. Gestione connessioni:

- Gestisce connessioni TCP e UDP per comunicare con il server.

### Server

1. Salvataggio dei dati:
   - La classe `DataStore` salva periodicamente i dati degli utenti e degli hotel in file JSON, garantendo che le informazioni siano persistenti.
2. Gestione degli hotel:
   - La classe `HotelManager` permette di cercare hotel, visualizzare tutti gli hotel di una città, inserire recensioni e notificare i client quando cambia il miglior hotel in una città.
3. Gestione degli utenti:
   - La classe `UserManager` permette di registrare nuovi utenti, effettuare il login e il logout, mostrare i badge degli utenti e incrementare il conteggio delle recensioni.
4. Elaborazione delle richieste dei client:
   - La classe `WorkerThread` gestisce le connessioni dei client, legge le richieste e utilizza le classi manager per fornire le risposte appropriate.

## Struttura della classe e dei metodi

### Client

- InputValidator
  - `isValidUsername()`: controlla la validità dello username.
  - `isValidPassword()`: controlla la validità della password.
  - `isValidCity()`: controlla la validità della città (una dei 20 capoluoghi di regione italiani).
  - `isValidScore()`: controlla la validità dei punteggi globali e singoli.
  - `isValidHotelName()`: controlla la validità del nome dell'hotel.
- ClientHelper
  - `sendRequest()`: Invia richieste al server e riceve risposte.

    - `close()`: Chiude i flussi di input/output.
- HOTELIERClientMain
    - `start()`: Avvia il client, stabilisce connessione al server e mostra il menu.
        - `parseConfigFileClient()`: Legge e imposta parametri di configurazione da un file JSON.
        - `register()`, `login()`, `logout()`: Gestiscono la registrazione, il login e il logout degli utenti.
        - `searchHotel()`, `searchAllHotels()`, `insertReview()`,`showMyBadges()`: Gestiscono la ricerca di hotel e l'inserimento di recensioni.
        - `openReceiverUDPConnection()`, `closeUDPConnection()`: Gestiscono le connessioni UDP.
        - `exit()`: Chiude la connessione al server e termina il client.

## Server

- DataStore
    - `saveData()`: Salva i dati degli utenti e degli hotel.
    - `saveUsers()`: Salva i dati degli utenti.
    - `saveHotels()`: Salva i dati degli hotel.
    - `fillTempHashMap()`: Riempie una mappa temporanea con gli hotel dal file JSON.
    - `mergeHotels()`: Unisce gli hotel dalla mappa temporanea con quelli esistenti.
    - `getHotelsArray()`: Converte la mappa degli hotel in un array.
- HotelManager
    - `searchHotel()`: Cerca un hotel per nome e città.
    - `searchAllHotels()`: Cerca tutti gli hotel in una città.
    - `insertReview()`: Inserisce una recensione per un hotel e aggiorna le informazioni dell'hotel.
    - `sendUDPNotificationToClient()`: Invia una notifica UDP al client.
    - `getCityHotels()`: Ottiene tutti gli hotel di una città dal file JSON.

    - `findHotelInCity()`: Trova un hotel nella mappa degli hotel.
    - `findHotelInJson()`: Trova un hotel nel file JSON.
    - `checkIfPresentAndAdd()`: Controlla se un hotel è già presente nella lista e lo aggiunge se non lo è.
- UserManager
    - `register()`: Registra un nuovo utente.
    - `login()`: Effettua il login di un utente.
    - `logout()`: Effettua il logout di un utente.
    - `showMyBadges()`: Mostra i badge di un utente.
    - `addReviewCount()`: Aumenta il conteggio delle recensioni di un utente e aggiunge un badge se necessario.
    - `isLoggedIn()`: Verifica se un utente ha effettuato il login.
    - `addBadge()`: Aggiunge un badge a un utente in base al numero di recensioni.
- WorkerThread
    - `run()`: Esegue il thread per gestire le richieste dei client.
    - `handleRequest()`: Gestisce le richieste dai client, esegue l'azione appropriata e invia la risposta.
- HOTELIERServerMain
    - `start()`: Accetta connessioni e gestisce i thread dei client.
    - `shutdownServer()`: Chiude correttamente il server.
    - `setupShutdownHook()`: Imposta il hook per la chiusura del server.
    - `loadUsers()`: Carica dati degli utenti da JSON.
    - `parseConfigFileServer()`: Parsa il file di configurazione.

## Algoritmo di Ranking

L'algoritmo di ranking, implementato nella classe `RankingAlgorithm`, è progettato per calcolare il punteggio di ranking di un hotel basandosi su diverse metriche derivate dalle recensioni degli utenti. Ecco un'analisi dettagliata dell'algoritmo:

1. Costante Lambda:

- La costante `LAMBDA` è utilizzata nella formula di decadimento esponenziale per ridurre il peso delle recensioni più vecchie.

2. Calcolo del Punteggio di Recenza (Currency Score):
   - Il metodo `calculateRanking` inizia calcolando il punteggio di recenza, iterando su tutte le recensioni di un hotel.
   - Per ogni recensione, calcola i giorni trascorsi dalla data della recensione corrente al tempo attuale.
   - Utilizza una funzione di decadimento esponenziale per attenuare il contributo delle recensioni più vecchie, sommando il risultato al `currencyScore`.

```
private static final double LAMBDA = 0.01;
double currencyScore = 0;

for(Review review : hotel.getReviews()) {
      double daysSinceReview = (now - review.getDate()) /1000/60/60/24;
      currencyScore += review.getGlobalScore() * Math.exp(-LAMBDA *
daysSinceReview);
}
```

3. Calcolo del Punteggio Medio (Medium Score):
   - Il metodo `calculateMediumScore` calcola il punteggio medio di un hotel.
   - Somma i punteggi individuali delle recensioni (punteggi globali, di pulizia, di posizione, di servizi e di qualità) e calcola la media

```
public static double calculateMediumScore(Hotel hotel) {
      double totalScore = 0;
      int reviewCount = hotel.getReviews().size();
      for(Review review : hotel.getReviews()) {
              double reviewScore = review.getGlobalScore();
              reviewScore += review.getCleaningScore();
              reviewScore += review.getPositionScore();
              reviewScore += review.getServicesScore();
              reviewScore += review.getQualityScore();
              totalScore += reviewScore / 5;
```

```
        }
        return totalScore / reviewCount;
    }
```

4. Calcolo del Punteggio di Quantità (Quantity Score):
    - Viene calcolato il punteggio di quantità utilizzando il logaritmo naturale del totale delle recensioni più uno, per evitare problemi con il logaritmo di zero.

```
double quantityScore = Math.log(hotel.getTotalRatings() + 1);
```

5. Combinazione dei Punteggi:
    - I tre punteggi (`currencyScore`, `mediumScore`, `quantityScore`) sono moltiplicati per ottenere il punteggio di ranking finale.
    - Il punteggio di ranking viene quindi impostato nell'oggetto `Hotel`.

```
return currencyScore * mediumScore * quantityScore;
```

6. Aggiornamento dell'Hotel:
    - Il punteggio medio e il punteggio di ranking calcolati vengono impostati nelle proprietà dell'hotel.

```
hotel.setRate(mediumScore);
hotel.setRankingRate(currencyScore * mediumScore * quantityScore);
```

Questo algoritmo combina la freschezza delle recensioni, la qualità media delle recensioni e la quantità delle recensioni per fornire un ranking che riflette in modo comprensivo la reputazione di un hotel nel tempo.

## Dettagli dei Metodi Fondamentali

### `sendRequest(String request)`

**Descrizione:**

- Invia una richiesta al server e attende la risposta.

### Parametri di ingresso:

- `request`: La richiesta da inviare al server in formato stringa.

### Parametri di uscita:

- Restituisce la risposta dal server come una stringa.

### Logica interna e flusso di esecuzione:

1. Invia la richiesta al server utilizzando un oggetto `PrintWriter`.
2. Legge la risposta dal server tramite un oggetto `BufferedReader`, accumulando le righe di risposta in un `StringBuilder` fino a quando non viene ricevuto un terminatore di linea vuoto.
3. Restituisce la risposta completa come stringa.

---

## `saveUsers()`

### Descrizione:

- Salva gli utenti aggiornati nel file JSON.

### Parametri di ingresso:

- Nessuno.

### Parametri di uscita:

- Nessuno.

### Logica interna e flusso di esecuzione:

1. Utilizza l'oggetto `Gson` per convertire la lista degli utenti `users` in formato JSON.
2. Scrive il JSON risultante nel file specificato `usersPath` utilizzando un oggetto `FileWriter`.

---

## `saveHotels()`

### Descrizione:

- Salva gli hotel aggiornati nel file JSON.

### Parametri di ingresso:

- Nessuno.

### Parametri di uscita:

- Nessuno.

### Logica interna e flusso di esecuzione:

1. Crea una mappa temporanea `tempHotels` per gli hotel e un ArrayList temporaneo `temp` per gli hotel da aggiungere.
2. Riempie la mappa temporanea utilizzando `fillTempHashMap()` e unisce gli hotel con quelli esistenti tramite `mergeHotels()`.
3. Converte la mappa di hotel in un array utilizzando `getHotelsArray()`.
4. Utilizza Gson per convertire l'array di hotel in formato JSON e lo scrive nel file specificato `hotelsPath` utilizzando un oggetto `FileWriter`.

---

## `searchHotel(String hotel, String city)`

### Descrizione:

- Cerca un hotel per nome e città.

### Parametri di ingresso:

- `hotel`: Nome dell'hotel da cercare.
- `city`: Città in cui cercare l'hotel.

### Parametri di uscita:

- Restituisce le informazioni dell'hotel trovato come stringa o un messaggio se l'hotel non è stato trovato.

### Logica interna e flusso di esecuzione:

1. Cerca l'hotel nella mappa degli hotel per città con `findHotelInCity()`.
2. Se l'hotel non è presente nella mappa, cerca nell'array JSON con `findHotelInJson()`.
3. Se l'hotel viene trovato, aggiornalo nella mappa degli hotel e restituisci le sue informazioni come stringa.
4. Altrimenti, restituisci un messaggio di "Hotel non trovato".

---

## `searchAllHotels(String city)`

### Descrizione:

- Cerca tutti gli hotel in una città specifica.

### Parametri di ingresso:

- `city`: Città di cui cercare gli hotel.

### Parametri di uscita:

- Restituisce le informazioni di tutti gli hotel trovati nella città specificata come stringa.

### Logica interna e flusso di esecuzione:

1. Utilizza un JsonReader per leggere gli hotel dal file JSON (`HOTELS_PATH`).
2. Filtra gli hotel per città e li aggiunge alla mappa degli hotel con `checkIfPresentAndAdd()`.
3. Ordina gli hotel per ranking e costruisce una stringa contenente le informazioni degli hotel trovati.

---

`insertReview(String hotelName, String cityName, Float globalScore, Map<String, Float> singleScores, User user)`

### Descrizione:

- Inserisce una recensione per un hotel, aggiorna le informazioni dell'hotel e gestisce le notifiche.

### Parametri di ingresso:

- `hotelName`: Nome dell'hotel in cui inserire la recensione.
- `cityName`: Città dell'hotel.
- `globalScore`: Punteggio globale della recensione (da 0 a 5).
- `singleScores`: Mappa dei punteggi singoli della recensione (chiave: aspetto della recensione, valore: punteggio da 0 a 5).
- `user`: Utente che inserisce la recensione.

### Parametri di uscita:

- Restituisce un messaggio di conferma dell'inserimento della recensione.

### Logica interna e flusso di esecuzione:

1. Cerca l'hotel nella mappa degli hotel per città con `findHotelInCity()` e, se necessario, nel file JSON con `findHotelInJson()`.
2. Aggiorna il punteggio e i punteggi singoli dell'hotel con i valori della nuova recensione.
3. Calcola il nuovo ranking dell'hotel
4. Gestisce le notifiche se l'hotel diventa il miglior hotel della città.
5. Incrementa il conteggio delle recensioni dell'utente con `UserManager.addReviewCount()`.

---

`handleRequest(Socket clientSocket)`

## Descrizione:

- Gestisce una richiesta ricevuta dal client tramite il socket specificato.

## Parametri di ingresso:

- `clientSocket`: Il socket del client che ha effettuato la richiesta.

## Parametri di uscita:

- Nessuno.

## Logica interna e flusso di esecuzione:

1. Stampa l'indirizzo IP del client che ha effettuato la richiesta.
2. Verifica se il socket del client è nullo o chiuso. Se è così, stampa un messaggio e termina il metodo.
3. Ottiene flussi di input (`BufferedReader`) e output (`PrintWriter`) dal socket del client.
4. Legge la richiesta inviata dal client tramite il flusso di input.
5. Se la richiesta è nulla, chiude il socket del client e stampa un messaggio di chiusura della connessione.
6. Suddivide la richiesta in parti utilizzando uno spazio come delimitatore.
7. Determina il tipo di richiesta in base alla prima parte della richiesta (`requestType`).
8. Crea istanze di UserManager e HotelManager per gestire le operazioni relative agli utenti e agli hotel.
9. Esegue operazioni specifiche in base al tipo di richiesta:
    - `REGISTER`: Registra un nuovo utente con il nome utente e la password forniti.
    - `LOGIN`: Esegue il login di un utente con il nome utente e la password forniti.
    - `LOGOUT`: Esegue il logout di un utente con il nome utente fornito.

- `SEARCH_HOTEL`: Cerca un hotel per nome e città utilizzando `HotelManager`.
- `SEARCH_ALL_HOTELS`: Cerca tutti gli hotel in una città specificata utilizzando `HotelManager`.
- `INSERT_REVIEW`: Inserisce una recensione per un hotel specificato utilizzando `HotelManager`.
- `SHOW_MY_BADGES`: Mostra i badge di un utente con il nome utente fornito.
- `Default`: Se la richiesta non corrisponde a nessun tipo valido, invia un messaggio di "Richiesta non valida" al client.

10. Invia la risposta al client utilizzando il flusso di output.

# Thread Utilizzati

## Lato Server

1. `Worker Thread`
   - Creato dinamicamente per ogni connessione accettata dal server socket principale.
   - Gestisce le richieste del client associato.
2. `DataStore Thread`
   - Thread per il salvataggio periodico dei dati su disco.
   - Gestisce il salvataggio dei dati degli utenti e degli hotel in file JSON.

## Lato Client

1. `UDP Receiver Thread`
   - Thread separato per ricevere messaggi UDP dal server.
   - Continuamente in ascolto per aggiornamenti dal server.

# Compilazione ed Esecuzione

## Compilazione

Per compilare tutti i file del progetto, è necessario raccogliere tutti i file Java presenti nella directory `./src` e salvarne i

percorsi in un file `sources.txt`. Successivamente, si utilizza il comando `javac` per compilare i file Java specificando la libreria esterna `gson-2.11.0.jar` e la directory di output `./bin`. Ecco i comandi necessari:

```
## CREAZIONE FILE SOURCES.TXT

find ./src -name "*.java" > sources.txt



## COMPILAZIONE DI TUTTI I FILE

javac -cp ./src/libs/gson-2.11.0.jar @sources.txt -d ./bin
```

## Creazione file.jar

Una volta completata la compilazione, si procede con la creazione dei file JAR eseguibili per il client e il server. Per ciascuno di essi, si utilizza il comando `jar` specificando il file manifest e la directory binaria contenente i file compilati. Ecco i comandi per creare i JAR:

```
## CREARE JAR CLIENT

jar cfm HOTELIERClientMain.jar manifestClient.txt -C bin .



## CREARE JAR SERVER

jar cfm HOTELIERServerMain.jar manifestServer.txt -C bin .
```

## Esecuzione jar

Dopo aver creato i file JAR, è possibile eseguire il server e il client utilizzando il comando `java -cp`, specificando il file JAR

da eseguire insieme alla libreria `gson-2.11.0.jar`. Ecco i comandi per eseguire il server e il client:

```
## JAR SERVER RUN

java -cp HOTELIERServerMain.jar:./src/libs/gson-2.11.0.jar
src.Server.HOTELIERServerMain



## JAR CLIENT RUN

java -cp HOTELIERClientMain.jar:./src/libs/gson-2.11.0.jar
src.Client.HOTELIERClientMain
```

Questi comandi garantiscono che il server e il client possano essere eseguiti correttamente, caricando le librerie necessarie e avviando le rispettive applicazioni.