Esercitazione 6 – JDBC

- Usare gli strumenti di Mysql per eseguire lo script mysqlScript (fornito dal docente)
- Nel progetto creare il package `database` da popolare con le classi: `DbAccess`, `TableData`, `TableSchema`, `DatabaseConnectionException`, `NoValueException`, `QUERY_TYPE` (fornite dal docente)

Classe DBAccess . (fornita dal Docente)

Attributi

`private final String DRIVER_CLASS_NAME = "com.mysql.cj.jdbc.Driver";` Per utilizzare questo Driver scaricare e aggiungere al classpath il connettore mysql connector)

`private final String DBMS = "jdbc:mysql";`

`private final String SERVER = "localhost";` contiene l'identificativo del server su cui risiede la base di dati (per esempio localhost)

`private final int PORT = 3306;` La porta su cui il DBMS MySQL accetta le connessioni

`private final String DATABASE = "Map";` contiene il nome della base di dati

`private final String USER_ID = "Student";` contiene il nome dell'utente per l'accesso alla base di dati

`private final String PASSWORD = "map";` contiene la password di autenticazione per l'utente identificato da USER_ID

`private Connection conn;` gestisce una connessione

Metodi

public void initConnection() throws DatabaseConnectionException: impartisce al class loader l'ordine di caricare il driver mysql, inizializza la connessione riferita da *conn*.

public Connection getConnection(): restituisce *conn*;

public void closeConnection(): chiude la connessione *conn*;

- Classe Table_Schema (fornita dal docente) che modella lo schema di una tabella nel database relazionale

- Classe NoValueException (fornita dal docente) che estende Exception per modellare l'assenza di un valore all'interno di un resultset

- Casse Table_Data fornita dal docente) che modella ll'insieme di tuple collezionate in una tabella. La singola tupla è modellata dalla classe Tuple_Data inner class di Table_Data.

Metodi

public List<TupleData> getTransazioni(String table) throws SQLException

Input: nome della tabella nel database.

Output: Lista di tuple memorizzate nella tabella.

Comportamento: Ricava lo schema della tabella con nome table. Esegue una interrogazione per estrarre le tuple da tale tabella. Per ogni tupla del resultset, si crea un oggetto, istanza della classe Tupla, il cui riferimento va incluso nella lista da restituire. In particolare, per la tupla corrente nel resultset, si estraggono i valori dei singoli campi (usando getFloat() o getString()), e li si aggiungono all'oggetto istanza della classe Tupla che si sta costruendo.

public List<Object> getDistinctColumnValues (String table, Column column) throws SQLException

Input: Nome della tabella, nome della colonna nella tabella

Output: Lista di valori distinti ordinati in modalità ascendente che l'attributo identificato da nome *column* assume nella tabella identificata dal nome *table*.

Comportamento: Formula ed esegue una interrogazione SQL per estrarre i valori distinti ordinati di *column* e popolare una lista da restituire.

public Object getAggregateColumnValue(String table, Column column,QUERY_TYPE aggregate) throws SQLException, NoValueException

Input: Nome di tabella, nome di colonna , operatore SQL di aggregazione (min,max)

Output: Aggregato cercato.

Comportamento: Formula ed esegue una interrogazione SQL per estrarre il valore aggregato (valore minimo o valore massimo) cercato nella colonna di nome *column* della tabella di nome

table. Il metodo solleva e propaga una NoValueException se il resultset è vuoto o il valore calcolato è pari a null

N.B. aggregate è di tipo QUERY_TYPE dove QUERY_TYPE è la classe enumerativa fornita dal docente

```
public enum QUERY_TYPE {

MIN, MAX

}
```

Rimpiazzare il costruttore della classe **Data** con un costruttore che si occupa di caricare i dati di addestramento da una tabella della base di dati. Il nome della tabella è un parametro del costruttore. (Classe fornita dal Docente)

Creare due progetti distinti, Client e Server.

SERVER (Include tutti i package/classi definiti finora a esclusione di MainClass)

- Definire il package server che conterrà le classi MultiServer e ServerOneClient.
- Definire la classe MultiServer che modella un server in grado di accettare la richiesta trasmesse da un generico Client e istanzia un oggetto della classe ServerOneClient che si occupera di servire le richieste del client in un thred dedicato. Il Server sarà registrato su una porta predefinita (al di fuori del range 1-1024), per esempio 8080.

  Attributi

  `private public static final int PORT = 8080;`

  Metodi

  `public static void main(String[] args)`: crea un oggetto istanza di MultiServer.

  `MultiServer()`: invoca il metodo privato run.

  `private void run()`: assegna ad una variabile locale `s` il riferimento ad una istanza della classe `ServerSocket` creata usando la porta `PORT`. `s` si pone in attesa di richieste di connessione da parte di client in risposta alle quali viene restituito l'oggetto `Socket` da passare come argomento al costruttore della classe `ServerOneClient`.

N.B. definire la classe MultiServer adattando l'esempio visto durante la lezione

- Definire la classe ServerOneClient che estende la classe Thread che modella la comunicazione con un unico client.

  Attributi

  `private Socket socket`: Terminale lato server del canale tramite cui avviene lo scambio di oggetti client-server

`private ObjectInputStream in:` flusso di oggetti in input al server.

`private ObjectOutputStream out:` flusso di oggetti in output dal server al client.

Metodi

`ServeOneClient(Socket socket) throws IOException:` Costruttore. Inizia il membro `this.socket` con il parametro in input al costruttore. Inizializza `in` e `out`, avvia il thread invocando il metodo `start()` (ereditato da `Thread`).

```
ServeOneClient(Socket s) throws IOException {
        socket = s;
        in  = new ObjectInputStream(s.getInputStream());
        ...
}
```

`public void run():` Ridefinisce il metodo `run` della classe `Thread` (variazione funzionale). Gestisce le richieste del client (apprendere pattern/regole e popolare con queste `archive`; salvare `archive` in un file, avvalorare `archive` con oggetto serializzato nel file)

(Implementare questo metodo tenendo conto della implementazione del client fornita dal docente)

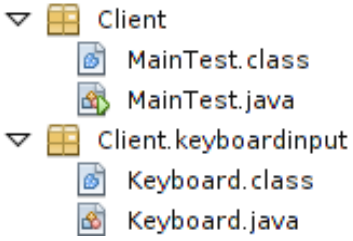
ESEMPIO DI OUTPUT

Server avviato

Connessione di Socket[addr=/127.0.0.1,port=40126,localport=2005]

Nuovo client connesso

CLIENT (Include MainTest e Keyboard)



Il client contatta il server usando ip e numero di porta su cui il server è in ascolto. Una volta instaurata la connessione (canale di comunizaione con terminale socket lato cliente) il client

trasmette le sue richiste (con i relativi parametri) al server e ne aspetta la risposta.

- Definire la classe MainTest (fornita dal docente)

Metodi

```
public static void main(String[] args) throws IOException, ClassNotFoundException
```

- Crea l'oggetto InetAddress che modella l'indirizzo del Server in rete. Crea l'oggetto Socket che deve collegarsi a tale Server. Inizializza i flussi di oggetti in e out per la trasmissione/ricezione di oggetti a/da server. Interagisce con l'utente per capire se questi vuore caricare un risultato esistente su file o crearne uno nuovo. In entrambi i casi trasmette la relativa richiesta e i necessari parametri al server (per esempio, min sup, minGr, nome tabella target, nome tabella backgorund) al server e ne aspetta la risposta che sarà poi stampata video.

ESEMPIO DI OUTPUT

output.txt

-