|  |  |  |
| --- | --- | --- |
| **ITMD 411** | ***Intermediate Software Development*** | Lab 4 |

**PROJECT Bank record loan generation report 100 points**

**Objective** To write a program that creates a Loan analysis report from class objects created in lab #2.

***PROJECT DESCRIPTION***

Bank of IIT now needs your help in deciphering whom from its records should be exclusive to premium loans versus those offered micro or less premium loans.

Use a database to store then present Loan analysis information from your data BankRecords objects you worked on in prior labs.

***Project Details***

For this lab you will continue using your current project folder created for labs 2 & 3 and create 3 new source files. Files are as follows:

File **DbConnect:** to allow an object to connect / close a database connection.

File **Dao:** where Dao stands for Data Access Object. This will allow for database connectivity and CRUD (Create Read Update Delete) like operations. Ref: for CRUD-> [Tutorialspoint](https://www.tutorialspoint.com/jdbc/index.htm).

File **LoanProcessing:** acts as a driver file (i.e., includes main function) to call your database CRUD methods and create some resulting output.

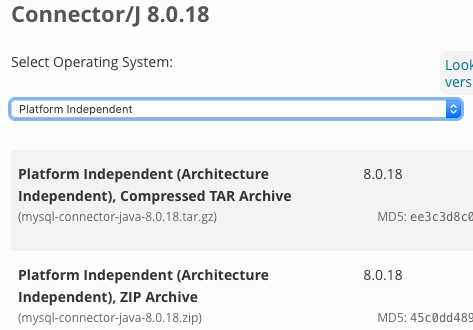
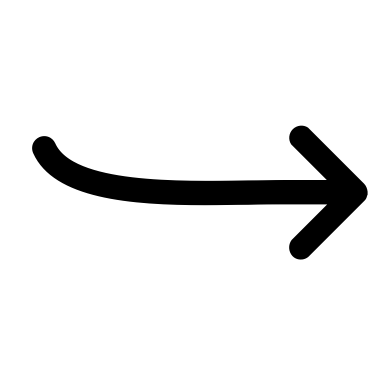
To start working with a database you will need a JDBC driver to allow for any connectivity within your app.

To include a driver for your MySQL database, you need to create a folder called **libs** within your project. The MySQL driver for JDBC connections packages in the form of a jar file which you can download here:

You will find a JDBC driver for connecting to a MySQL database located here:

<https://dev.mysql.com/downloads/connector/j/>

Depending on your OS you will need to download a windows zip file or a tar file if you are a Mac/Linux user. I usually just choose 'Platform Independent' for Mac/Windows from the pull down menu as shown next and then choose to download the **ZIP Archive** version.



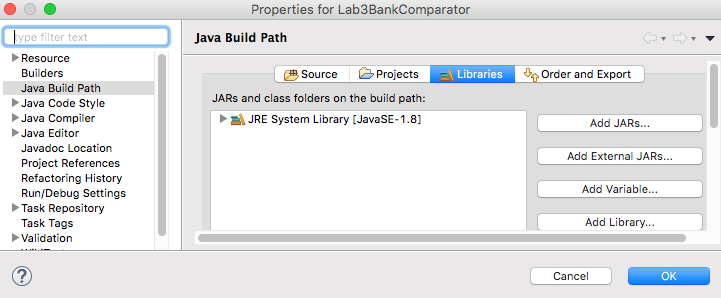
Click the  button given your choice at the right hand side, and you will be taken to a page to begin your download. Go to the bottom of the page and click on the

**No thanks, just start my download** link. Unzip or untar the downloaded file and locate the needed and "latest" .jar file ( mysql-connector-java-8.0.19.jar ) and copy it into into your **libs** folder.

Make sure to include the jar file in your Build Path by right clicking on your project folder and then choose **Build Path > Configure Build Path....**

Once the dialog box opens make sure to click on the **Libraries** tab then click

**Add JARs…**



Drill down to the libs folder within your project and then choose the jar file to add.

Click OK to commit. Then click Apply and Close. Snapshot follows.

A screenshot of a cell phone

Description automatically generated

Code work

Include the following helper code for your 3 files. (note- if some assignment statements in the spec continue to wrap to another line, you may want to adjust your source editor to include the assignment onto one line).

New file 1: DBConnect.java. Include the following for your new file.

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** DBConnect {

// Code database URL

**static** **final** String ***DB\_URL*** = "jdbc:mysql://www.papademas.net:3307/411labs?autoReconnect=true&useSSL=false";

// Database credentials

**static** **final** String ***USER*** = "db411", ***PASS*** = "411";

**public** Connection connect() **throws** SQLException {

**return** DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

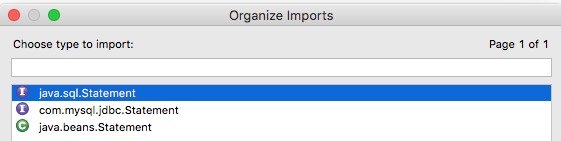
}

}

Remaining helper code below will not include imports for brevity, so please make sure to add them in (**Source > Organize Imports**)!

Also further note, when *choosing* imports for db operations from source classes, if prompted, for choosing from more than one option, choose from the **java.sql**, package and subpackages in the list.

Ex.



New file 1: Dao.java. . Include the following for your new file.

-Include the following class fields, constructor into your new file

//Declare DB objects

DBConnect conn = **null**;

Statement stmt = **null**;

// constructor

**public** Dao() { //create db object instance

conn = **new** DBConnect();

}

-Include a method to create a database table called **createTable**. createTable merely

creates a table when the method is called. Include the fields **pid**, **id**, **income** and **pep**

when building your table setup. A PRIMARY KEY which ensures record uniqueness is included for your build for the **pid** field which is shown below.

[ Note when creating a table, it is IMPERATIVE to include the following name:

**yourFirstinitial\_First4LettersOfYourLastName\_tab** ]

// CREATE TABLE METHOD

**public** **void** createTable() {

**try** {

// Open a connection

System.***out***.println("Connecting to a selected database to create Table...");

System.***out***.println("Connected database successfully...");

// Execute create query

System.***out***.println("Creating table in given database...");

stmt = conn.connect().createStatement();

String sql = "CREATE TABLE yourTableName\_tab " +

"(pid INTEGER not NULL AUTO\_INCREMENT, " +

" id VARCHAR(10), " +

" income numeric(8,2), " +

" pep VARCHAR(4), " +

" PRIMARY KEY ( pid ))";

stmt.executeUpdate(sql);

System.***out***.println("Created table in given database...");

conn.connect().close(); //close db connection

} **catch** (SQLException se) {

// Handle errors for JDBC

se.printStackTrace();

}

}

Notice carefully that comments are put in code as well the display of relevant information to the console. For future methods, continue this way (*note*- see snapshot at the last page).

-Include a method to insert records called **insertRecords()**.

// INSERT INTO METHOD

**public** **void** insertRecords(BankRecords[] robjs) {

**try** {

// Execute a query

System.***out***.println("Inserting records into the table...");

stmt = conn.connect().createStatement();

String sql = **null**;

// Include all object data to the database table

**for** (**int** i = 0; i < robjs.length; ++i) {

// finish string assignment to insert all object data

// (id, income, pep) into your database table

sql = " ";

stmt.executeUpdate(sql);

}

conn.connect().close();

} **catch** (SQLException se) { se.printStackTrace(); }

}

Finish coding the above sql string where commented with an insert statement (example insert statement follows):

      sql = "INSERT INTO yourTableName\_tab(field 1,field 2, field n) " +

       "VALUES (' "+value 1+" ', ' "+value 2+" ', ' "+value n+" ' )";

Note for brevity purposes, future starter code will EXCLUDE try / catch blocks. Add in your own try / catch blocks were applicable.

-Include a method to retrieve records for display called **retrieveRecords()**. Make to include try/catch blocks or a throws declaration when prompted. In like manner follow through with the same error trap logic when issues are presented to do so.

**public** ResultSet retrieveRecords() {

ResultSet rs = **null**;

stmt = conn.connect().createStatement();

String sql = "SELECT \* from yourTableName\_tab";

rs = stmt.executeQuery(sql);

conn.connect().close();

**return** rs;

}

**Methods breakdown**

insertRecords(BankRecords [] arrayName) will allow for the array of BankRecord objects, to be passed to your method which will allow for the insertion of all the **id**, **income** and **pep** data from your BankRecords array (or *whatever* you named it) via your getters, into your database table when called.

retrieveRecords() will return a **ResultSet** object used for creating output. The result set contains record data including your id, income and pep table fields.

\*Code tweak: Make sure to sort the pep field in *descending* order to allow for premium loan candidates to appear first in the record set for reporting purposes (i.e., those with data values of “YES”). The resultset query string to build can be something like:

String sql =

"select id,income, pep from yourTableName\_tab order by pep desc";

As a quick note: make sure to always close out of your connections and any statements when through with any processing!

Again, make sure to include error trapping using SQLException handling for all your database operations and connection logic.

Again, include messages to the console when your methods trigger. Ex. Table created, inserting records into database, etc.

A super great resource to assist you with all your JDBC-CRUD operations for your methods can be found at this site: <http://www.tutorialspoint.com/jdbc/>, as well as the Chapter 22 PowerPoint from Gaddis. Remember though to phrase coding the best you can using your own object names, method naming and variable names, including coding syntax and even comments, if referencing any material from tutorialspoint so your lab work is unique.

Lastly code your LoanProcessing class file as per the following instructions.

For your LoanProcessing source file make sure to extend **BankRecords**.

Include the following methods for your file.

Include in main(), your readData() method call which will process your BankRecord objects. Then instantiate a Dao object and trigger your createTable() method, your insertRecords(your BankRecords array object) method and retrieveRecords() method in that order. **Of course you can comment out your createTable / insertRecords once you’ve accomplished that to test how your output statements appear**. Logic for that follows next.

Once you have retrieved a recordset, print out all the records from the recordset to the console in a nice columnar format included with heading names for id, income and pep. Doesn’t hurt to have a title too, like Loan Analysis Report.

Sample starter code for **main** (fill in code where comments are shown to do so):

BankRecords br = **new** BankRecords();

br.readData();

Dao dao = **new** Dao();

dao.createTable();

dao.insertRecords(*robjs*); // perform inserts

ResultSet rs = dao.retrieveRecords(); // fill result set object

// Create heading for display

// Extract data from result set

**while** (rs.next()) {

// Retrieve data by column name (i.e., for id,income,pep)

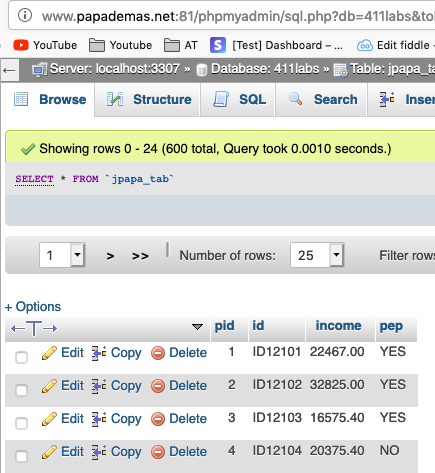
// Display values for id,income,pep

}

rs.close(); // closes result set object

Give a trial run at this point to set things in motion and observe your results. Always a good idea to check your table creation too in MySQLWorkbench or at [www.papademas.net:81/phpmyadmin](http://www.papademas.net:81/phpmyadmin)!

Sample successful table generation view from initial run follows.





Note- if you ever need to recreate your table, you can do so on your own, no problem. Unfortunately you cannot drop your table name / recortds as you do not have permission to do so. JUST BE SURE TO UPDATE YOUR TABLE NAMES WHEREVER APPLICABLE IN CODE.

Extra Credit options

-Include SQL Prepared statements when inserting records (**+5 points**)

-Include besides console output, a JTable GUI output display of your recordset

data (**+5 points**)

For a [JTable](https://docs.oracle.com/javase/tutorial/uiswing/components/table.html) to trigger code, build a frame with Javax [Swing](https://docs.oracle.com/javase/tutorial/uiswing/index.html) components which will

display your loan data into a JTable. Column headings must be included as well as a

scrollbar for proper appearance of your display within a GUI [frame](https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html) showing your table

component.

Or you for extra credit you may choose the following option

- Serialize and deserialize BankRecord objects using the java.util.Map class. (**+10 points**)

Name your serializable file *bankrecords.ser*. Make your application sleep for 5 seconds

between the processes. Display to the console the time difference between the serialization

and deserialization processes.

Include a **zip** file of all your project source code (new and old), plus your csv file, and a snapshot of your app at runtime which *must* include a table creation message, an insertion message and the first ten rows of your record results showing the Loan Analysis Report data into a **doc** file for credit.

If you have any extra credit, snapshot that as well and label your snapshot(s) accordingly. Sample display output follows…

Sample output display

