|  |  |  |
| --- | --- | --- |
| **ITMD 510** | ***Object Oriented App Development*** | Lab 4 |

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

**Objective** To write a program with a MVC ‘simulated’ approach that performs a Loan analysis 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 you will create the following packages & their containments in an MVC style as follows:

**Package**: **models**

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

**File** **DaoModel:** where Dao stands for Data Access Object, defines CRUD (Create Read Update Delete) like operations.

**Package**: **controllers**

**File** **LoanProcessing:** acts as a controller or “driver” file (i.e., includes main function) to run database CRUD methods and fires up some resulting output.

**Package**: **views**

**File LoanView:** shows JTable 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/>

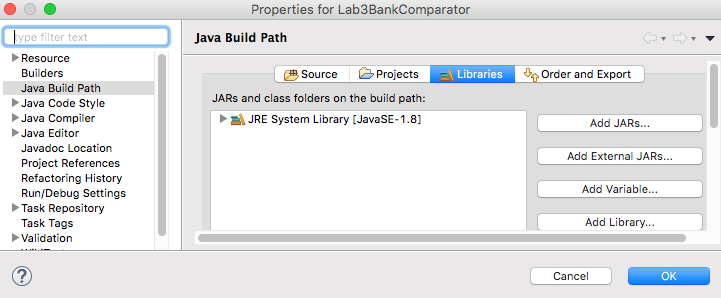
Depending on your OS you will need to download a windows zip file or a tar file if you are a Mac/Linux user. Click the  button given your choice, and you will be taken to a page to begin your download. Go to the bottom of the page and click on the

**No thanks, just start my download** link. Unzip or untar the downloaded file and locate the needed .jar file ( mysql-connector-java-5.1.46-bin.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.

Project work

Create the following packages and corresponding files and include the helper code provided as follows (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).

Okay for a slight workaround if necessary! **IF** you DO NOT have a package name for your project files thus far (i.e., it shows as a *default package*), let’s go ahead and create one to make it possible for importation needs so file objects can be used! So again if your current src folder shows as follows



you need to create a package and move all files from your default package into your new package. To add in a new package, right click on your source folder (src) and choose New > Package. Name your package **records**. Leave the default setting. Choose Finish to complete this step.

Now highlight all your files in your default package and move them (drag them over) into your new package. And voila, all your moved files will now have **package records;** automatically added to each file heading! Plus your default package is now blown away.

Note also if you already have had a package name for your project work, keep it as it will automatically be imported when time comes!

Now to continue with your code work, right click on your source folder (src) and choose New > Package. Name your package **models**. Leave the default setting. Choose Finish to complete the step.

Next, right click on your package and create a class. Name your class **DBConnect** and copy in the following code to allow connections to the papaserver.

DBConnect.java source

**package** models;

**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:3306/510labs?autoReconnect=true&useSSL=false";

// Database credentials

**static** **final** String ***USER*** = "db510", ***PASS*** = "510";

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

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

}

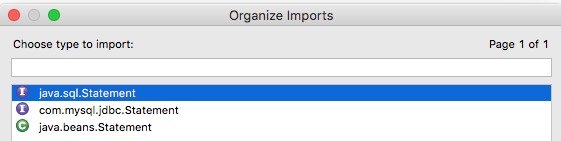
}

Note that any remaining helper code below will not include imports or any package names for brevity, so please make sure to add them in (Source > Organize Imports)! Note when you choose Organize Imports, automatically any imports for any of your new packages will automatically be included as well! ☺

Also note that if and when you copy code in, you may get some underlying errors, until you include any necessary import statements.

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.



In the models package add in a class named **DaoModel** and code it as follows:

DaoModel.java source

-Include the following class fields, constructor

//Declare DB objects

DBConnect conn = **null**;

Statement stmt = **null**;

// constructor

**public** DaoModel() { //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, " +

" id VARCHAR(10), " +

" income numeric(8,2), " +

" pep VARCHAR(3), " +

" 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.

-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();

// Include all object data to the database table

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

// finish string assignment to insert all object data

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

String sql = " ";

stmt.executeUpdate(sql);

}

conn.connect().close();

} **catch** (SQLException se) {

se.printStackTrace();

}

}

Finish coding the above sql string with an insert statement where commented.

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 mostly EXCLUDE try / catch blocks. Add in your own try / catch blocks were applicable.

-Include a method to retrieve records for display called **retrieveRecords()**.

**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) 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 pid, 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!

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 23 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.

Next in the same manner as you created your models package, create a package called **controllers** and include a class called **LoanProcessing** for your package. Code your class file as follows

LoanProcessing.java source

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

Simply follow thru with the following object creations and method calls in main.

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

br.readData();

DaoModel dao = **new** DaoModel();

dao.createTable();

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

ResultSet rs;

**try** {

rs = dao.retrieveRecords();

**new** LoanView().runView(rs);

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

-Note you will error out on the *extends* clause and with the

**new LoanView().runView(rs);** line of code added above, until you include appropriate imports and the LoanView class which will be performed next.

**Main code breakdown**

Included in main() is your readData() method call which will process your BankRecord objects. Then instantiate a DaoModel object and trigger your createTable() method, your insertRecords(your BankRecords array object) method and your 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 shortly.

Once you have retrieved a recordset, you will call a **runView** method from the **LoanView** class to print out all the records from the recordset to a Java window which contains a JTable in a nice columnar format included with heading names for pid, id, income and pep and a nice title for the report!

Lastly now create a package named **views** and include a class called **LoanView**. Code LoanView’s runView method as follows

LoanView.java source

**public** **void** runView(ResultSet rs) {

// instantiate vector objects to hold column/row data for JTable

Vector<Vector<Object>> data = **new** Vector<Vector<Object>>();

Vector<String> column = **new** Vector<String>();

**try** {

ResultSetMetaData metaData = rs.getMetaData();

**int** columns = metaData.getColumnCount();

// get column names from table!

String cols = "";

**for** (**int** i = 1; i <= columns; i++) {

cols = metaData.getColumnName(i);

column.add(cols);

}

// get row data from table!

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

Vector<Object> row = **new** Vector<Object>(columns);

**for** (**int** i = 1; i <= columns; i++)

row.addElement(rs.getObject(i));

data.addElement(row);

}

DefaultTableModel model = **new** DefaultTableModel(data, column);

JTable table = **new** JTable(model);

JFrame frame = **new** JFrame("Loan Details");

frame.setSize(700, 200);

frame.add(**new** JScrollPane(table));

frame.setDefaultCloseOperation(0);

frame.pack();

frame.setVisible(**true**);

rs.close(); //close ResultSet instance

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

}

**Code breakdown**

To view the output in a nice table view, a [JTable](https://docs.oracle.com/javase/tutorial/uiswing/components/table.html) object is created to take in rows and columns (column names retrieved from generated metadata) from your ResultSet (passed through the runView method’s parameter) which stores the data each as a Vector which becomes displayed in a [JFrame](https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html) which is part of Java’s [Swing](https://docs.oracle.com/javase/tutorial/uiswing/index.html) API.

Now before running your code from your LoanProcessing main file, go back into your LoanProcessing file and import the needed package/class name to clear your error as noted before.

Extra Credit options

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

-Show console output of Loan Analysis Report detailed data + a report title (**+5 points**)

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 a jar file and a snapshot of your console at runtime which *must* include a table creation message, an insertion message and a snapshot of your JTable showing the first few rows of your record results into a doc file into BB for credit. If you have any extra credit, snapshot that as well and label your extra snapshot(s) accordingly. Sample displays of output follows…

Sample runs

