**Appendices L**

**Database Development – Java Database Connectivity (JDBC)**

1. **Connecting to the Database**

To implement a connection to the MySQL database from the Database class, I had to amend the Database constructor to pass MySQL database connection information (IP address, username, password, and port number), along with the environment the system is, i.e. LIVE, TRAIN or TEST.

Now the Database class has the connection information, the constructor invokes a local method called connect() and passes the information as parameters.

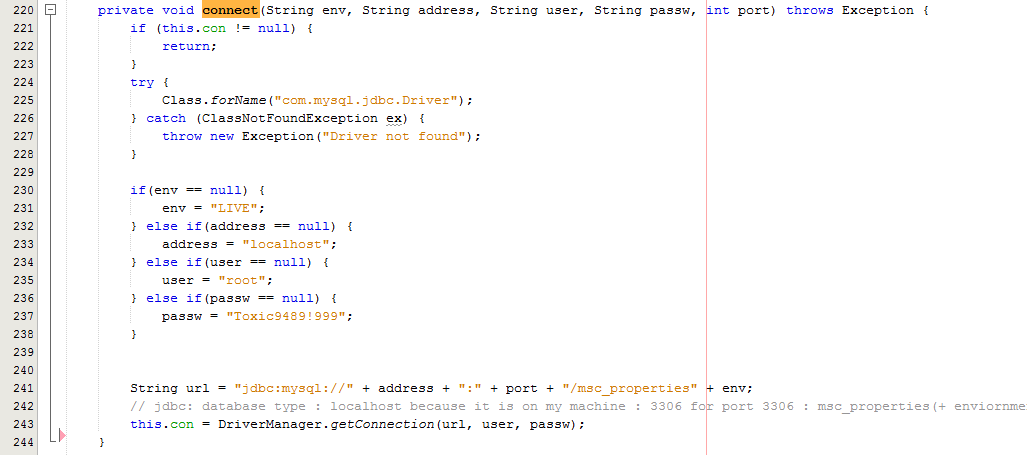


Fig. 1 – Database class – connect()

Firstly, as you can see from Fig. 1, I use the singleton pattern to ensure that the variable Connection con that is defined within the Database class is null, and if not then I don’t execute the main try statement, which creates the connection and assigns it to the con variable.

If there is no connection open already, we execute a try statement, which invokes the static method Class.forName(), providing the String value “com.mysql.jdbc.Driver” as a parameter. The Class.forName() method dynamically loads the JDBC driver which will enable the Database class to interact with the MySQL database.

I then constructed a String value called url consisting of the API type, the database type, the IP address of the database, the port number and the database name, which for ‘MSc Properties’ consists of msc\_properties + the environment, for example msc\_propertiesLIVE. Once the string is constructed I then invoked the static method DriverManager.getConnection() and pass the String url, and the database username and password as parameters to the getConnection() method, the getConnection() method then returns a Connection object, which I assign to the con variable. I am then able to invoke methods on the con variable to interact with the MySQL database.

1. **Loading System Data at Start-up (Read)**

To be able to load system data at system start up, was quite a tricky task as I had to ensure that no objects was loaded up prior to an object that the loaded object is dependent on was loaded, and also needed to ensure all system elements such as title codes, religion codes was loaded up first.

Once I had mapped out a flow in which I could load data from the MySQL database and create objects from that data without causing any issues with dependent data not being available, I was then able to create a load method which will create the system objects within the database class at system start up.

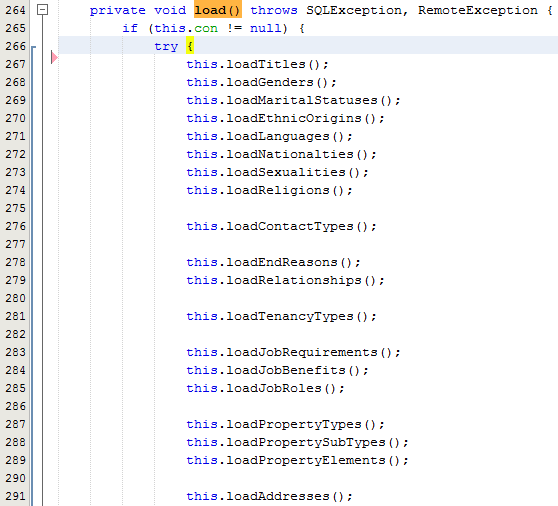


Fig. 2 – Extract from Database class - load() part 1

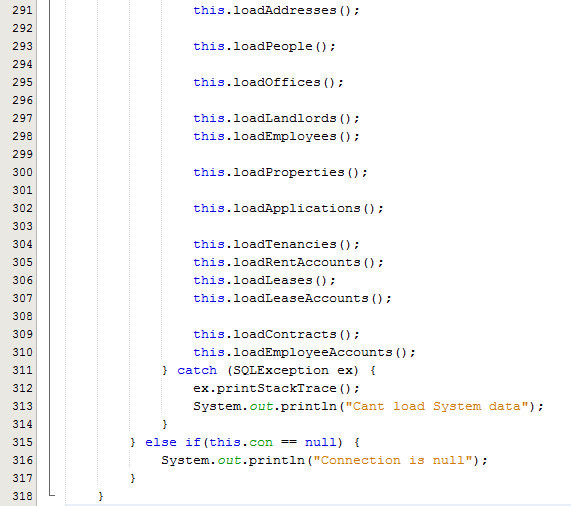


Fig. 3 – Extract from Database class - load() part 2

Once the load method in Fig. 2 and Fig. 3 was created I then had to implement the individual load methods which will deal with loading sets of records from the MySQL database, create the objects and adding them to the Lists within the Database class.



Fig. 4 – Extract from Database class - loadAddresses()

As you can see from Fig. 4, the loadAddresses() method shows that firstly I had to create a String called sql, which will be the sql select statement I want to execute to retrieve the address records from the database. I then execute a try with resources statement, which declares one or more resources [], I declare a Statement variable called statement, and assign it the return value from invoking createStatement() on the con variable for the Database class within this try with resources statement.

Once I have the Statement variable initialized I am then able to invoke executeQuery() on the Statement variable and pass the String sql as a parameter, this will return a ResultSet object which contains the returned data from executing the select statement, the ResultSet object is then assigned to a ResultSet variable I declared called results.

Now I have the returned results I use a while loop with the condition being the return value of invoking next() on the ResultSet variable, which returns true if there is another record to return. So if there is another record in the ResultSet variable, I then invoke a get method to return a piece of data depending on the column name given as the methods parameter. There are a number of get methods such as getInt(), getDate, getString etc. to return all different data types.

Once all of the records information has been retrieved I then create the required object, in this example I had to create a Note object, which is a parameter for the creation of the Address object, once the objects have been created, I add them to the lists within the Database class. Once the method has finished I then need to close the statement by invoking close() on the Statement variable.

1. **Create, Update and Delete Data**

Although I am not going to talk about creating, updating and deleting records from the MySQL database in as much detail, as the tasks are similar I will show you a brief example of each.



Fig. 5 – Extract from Database class, createAddress()

In Fig. 5, I am creating an insert statement, a PreparedStatement variable called insertStat, which means I can then add the information to the PreparedStatement through the use of set methods being invoked on the PreparedStatement and using the ? placeholder for the values I am going to supply. Once all of the information has been assigned using the set methods, the executeUpdate() method needs to be invoked and then the connection needs to be closed through close(). Once the connection is closed, I then add the objects to their respective lists within the database class.



Fig. 6 – Extract from Database class, updateAddress()

In Fig. 6, I am creating an update statement, and as with the insert statement I use a PreparedStatement to supply the update values, use the set methods to set the values, executeUpdate and then close the connection.

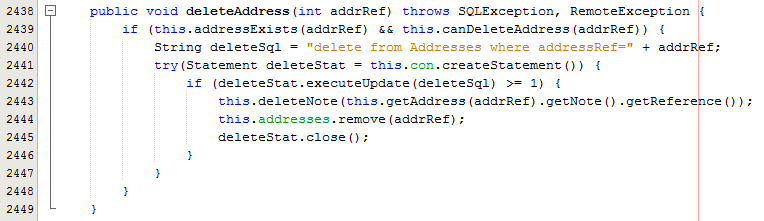


Fig. 7 – Extract from Database class, deleteAddress()

In Fig. 7, I am creating an update statement, and as with the load methods I use a Statement, and just invoke the executeUpdate method on the statement and supply the deletSql String value as a parameter. I then remove the object from the List within the Database class and close the connection.

The importance of implementing the MySQL database is that if the system crashes or needs to be shut down (over the periods ‘MSc Properties’ is closed), there needs an external storage outside of the system to store the information of the system, and then when the system starts up, we are able to access this information as I have shown above to bring the state of the system back to what it was prior to shut down or system crash.