Appendices K

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.

```
220     private void connect(String env, String address, String user, String passw, int port) throws Exception {
221         if (this.con != null) {
222             return;
223         }
224         try {
225             Class.forName("com.mysql.jdbc.Driver");
226         } catch (ClassNotFoundException ex) {
227             throw new Exception("Driver not found");
228         }
229 
230         if(env == null) {
231             env = "LIVE";
232         } else if(address == null) {
233             address = "localhost";
234         } else if(user == null) {
235             user = "root";
236         } else if(passw == null) {
237             passw = "Toxic9489!999";
238         }
239 
240 
241         String url = "jdbc:mysql://" + address + ":" + port + "/msc_properties" + env;
242         // jdbc: database type : localhost because it is on my machine : 3306 for port 3306 : msc_properties(+ enviornme
243         this.con = DriverManager.getConnection(url, user, passw);
244     }
```

Firstly, 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.

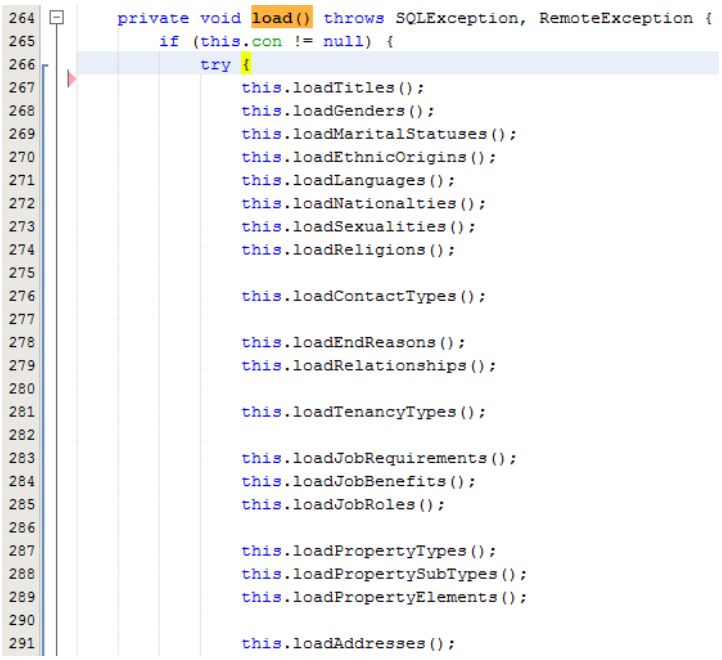
I then constructed a String value called url consisting of the API type, the database type, the IP address of the database, the port number and the database name, which for 'MSc Properties' consists of msc_properties + the environment, for example msc_propertiesLIVE. Once the string is constructed I 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.

2. 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.



```
264     private void load() throws SQLException, RemoteException {
265         if (this.con != null) {
266             try {
267                 this.loadTitles();
268                 this.loadGenders();
269                 this.loadMaritalStatuses();
270                 this.loadEthnicOrigins();
271                 this.loadLanguages();
272                 this.loadNationalties();
273                 this.loadSexualities();
274                 this.loadReligions();
275 
276                 this.loadContactTypes();
277 
278                 this.loadEndReasons();
279                 this.loadRelationships();
280 
281                 this.loadTenancyTypes();
282 
283                 this.loadJobRequirements();
284                 this.loadJobBenefits();
285                 this.loadJobRoles();
286 
287                 this.loadPropertyTypes();
288                 this.loadPropertySubTypes();
289                 this.loadPropertyElements();
290 
291                 this.loadAddresses();
```

```
291                 this.loadAddresses();
292
293                 this.loadPeople();
294
295                 this.loadOffices();
296
297                 this.loadLandlords();
298                 this.loadEmployees();
299
300                 this.loadProperties();
301
302                 this.loadApplications();
303
304                 this.loadTenancies();
305                 this.loadRentAccounts();
306                 this.loadLeases();
307                 this.loadLeaseAccounts();
308
309                 this.loadContracts();
310                 this.loadEmployeeAccounts();
311             } catch (SQLException ex) {
312                 ex.printStackTrace();
313                 System.out.println("Cant load System data");
314             }
315         } else if(this.con == null) {
316             System.out.println("Connection is null");
317         }
318     }
```

Fig x – Extract from Database class, load()

Once the above load method 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 add them to the Lists within the Database class.

```
2473    private void loadAddresses() throws SQLException, RemoteException {
2474        String sql = "select addressRef, buildingNumber, buildingName, subStreetNumber, subStreet, streetNumber, street, area, "
2475                + "town, country, postcode, noteRef, comment, createdBy, createdDate from addresses order by addressRef";
2476        try (Statement selectStat = con.createStatement()) {
2477            ResultSet results = selectStat.executeQuery(sql);
2478
2479            while (results.next()) {
2480                int addressRef = results.getInt("addressRef");
2481                String buildingNumber = results.getString("buildingNumber");
2482                String buildingName = results.getString("buildingName");
2483                String subStreetNumber = results.getString("subStreetNumber");
2484                String subStreet = results.getString("subStreet");
2485                String streetNumber = results.getString("streetNumber");
2486                String street = results.getString("street");
2487                String area = results.getString("area");
2488                String town = results.getString("town");
2489                String country = results.getString("country");
2490                String postcode = results.getString("postcode");
2491                int noteRef = results.getInt("noteRef");
2492                String comment = results.getString("comment");
2493                String createdBy = results.getString("createdBy");
2494                Date createdDate = results.getDate("createdDate");
2495
2496                Note note = new NoteImpl(noteRef, comment, createdBy, createdDate);
2497                Address temp = new Address(addressRef, buildingNumber, buildingName, subStreetNumber,
2498                        subStreet, streetNumber, street, area, town, country, postcode, note, createdBy, createdDate);
2499                this.addresses.put(temp.getAddressRef(), temp);
2500                this.notes.put(note.getReference(), note);
2501                this.loadAddressMods(temp.getAddressRef(), this.loadModMap("addressModifications", temp.getAddressRef()));
2502            }
2503            selectStat.close();
2504        }
2505    }
```

Fig x – Extract from Database class

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.

3. 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.

```
2370    public void createAddress(Address address) throws SQLException, RemoteException {
2371        if(!this.addressExists(address.getAddressRef())) {
2372            String insertSql = "insert into addresses (addressRef, buildingNumber, buildingName, subStreetNumber, subStreet, "
2373                    + "streetNumber, street, area, town, country, postcode, noteRef, comment, createdBy, createdDate) "
2374                    + "values (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
2375            try (PreparedStatement insertStat = this.con.prepareStatement(insertSql)) {
2376                int col = 1;
2377                insertStat.setInt(col++, address.getAddressRef());
2378                insertStat.setString(col++, address.getBuildingNumber());
2379                insertStat.setString(col++, address.getBuildingName());
2380                insertStat.setString(col++, address.getSubStreetNumber());
2381                insertStat.setString(col++, address.getSubStreet());
2382                insertStat.setString(col++, address.getStreetNumber());
2383                insertStat.setString(col++, address.getStreet());
2384                insertStat.setString(col++, address.getArea());
2385                insertStat.setString(col++, address.getTown());
2386                insertStat.setString(col++, address.getCountry());
2387                insertStat.setString(col++, address.getPostcode());
2388                insertStat.setInt(col++, address.getNote().getReference());
2389                insertStat.setString(col++, address.getComment());
2390                insertStat.setString(col++, address.getCreatedBy());
2391                insertStat.setDate(col++, DateConversion.utilDateToSQLDate(address.getCreatedDate()));
2392                insertStat.executeUpdate();
2393                insertStat.close();
2394                this.addresses.put(address.getAddressRef(), address);
2395                this.notes.put(address.getNote().getReference(), address.getNote());
2396            }
2397        }
2398    }
```

Fig x – Extract from Database class, createAddress()

In Fig x, 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.

```
2406    public void updateAddress(int addressRef) throws SQLException, RemoteException {
2407        if(this.addressExists(addressRef)) {
2408            AddressInterface address = this.getAddress(addressRef);
2409            String updateSql = "update addresses set buildingNumber=?, buildingName=?, subStreetNumber=?, subStreet=?, "
2410                    + "streetNumber=?, street=?, area=?, town=?, country=?, postcode=?, comment=? where addressRef=?";
2411            try (PreparedStatement updateStat = con.prepareStatement(updateSql)) {
2412                int col = 1;
2413                updateStat.setString(col++, address.getBuildingNumber());
2414                updateStat.setString(col++, address.getBuildingName());
2415                updateStat.setString(col++, address.getSubStreetNumber());
2416                updateStat.setString(col++, address.getSubStreet());
2417                updateStat.setString(col++, address.getStreetNumber());
2418                updateStat.setString(col++, address.getStreet());
2419                updateStat.setString(col++, address.getArea());
2420                updateStat.setString(col++, address.getTown());
2421                updateStat.setString(col++, address.getCountry());
2422                updateStat.setString(col++, address.getPostcode());
2423                updateStat.setString(col++, address.getComment());
2424                updateStat.setInt(col++, address.getAddressRef());
2425                updateStat.executeUpdate();
2426                updateStat.close();
2427            }
2428            this.createModifiedBy("addressModifications", address.getLastModification(), address.getAddressRef());
2429        }
2430    }
```

Fig x – Extract from Database class, updateAddress()

In Fig x, 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.

```
2438    public void deleteAddress(int addrRef) throws SQLException, RemoteException {
2439        if (this.addressExists(addrRef) && this.canDeleteAddress(addrRef)) {
2440            String deleteSql = "delete from Addresses where addressRef=" + addrRef;
2441            try(Statement deleteStat = this.con.createStatement()) {
2442                if (deleteStat.executeUpdate(deleteSql) >= 1) {
2443                    this.deleteNote(this.getAddress(addrRef).getNote().getReference());
2444                    this.addresses.remove(addrRef);
2445                    deleteStat.close();
2446                }
2447            }
2448        }
2449    }
```

Fig x – Extract from Database class, deleteAddress()

In Fig x, 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.