Software

To finish this lab, you may need the following software:

1. NetBeans IDE version 12.2 or 12.5
2. JDK version 1.8.0 (jdk1.8.0_202, or later)
3. GlassFish Server Open Source Edition version 5.1.0
4. Java DB - the database server that comes with GlassFish

Aim

This Lab has two aims:

1. Writing JDBC program to set up database table and data for the GlassFish
2. Familiarize yourself with NetBeans IDE

Note: You will need to use this program in Lab_03_Entity_Class_JPA to create the required database table in the database server and populate some data to the database.

Overview of Lab Tasks

Make sure that you have set up Java DB server – see Lab_01a step 6.

In this Lab, you will learn to perform the following tasks in writing JDBC programs:

LT1. Create a normal Java Application project in NetBeans
LT2. Add an external library to the Java Application project
LT3. Create a Java class
LT4. Write a method that establishes the database connection via JDBC
LT5. Write a method that creates a database table via JDBC
LT6. Write a method that deletes a database table via JDBC
LT7. Create a Java class for storing the information
LT8. Write a method that add multiple records to a database table via JDBC
LT9. Using NetBeans to connect to the database table
LT10. Write a Java program to tie all the previous work together
LT11. Verify the results of program written in LT7 in NetBeans
LT12. Write a method that adds a record to a database table via JDBC

Lab Tasks

This Lab should be run on MS Windows Platform

LT1. In this Lab task, you will learn how to create a new "Java Application" Project in NetBeans called "ED-JDBC"

LT1.1. Select "File" > "New Project"
LT1.2. Choose "Java with Ant" > "Java Application", then click "Next"
LT1.3. Enter "`ED-JDBC`" in the "Project Name" field
LT1.4. Change the "Create Main Class" field to "`ed.jdbc.SetUpMyUser`" (Make sure that the check box is checked)
LT1.5. Click "Finish"

Note 1: The new project "`ED-JDBC`" will be created in the "Projects" Windows.
Note 2: In the editor windows, the file "SetUpMyUser.java" has been created for you. We do not need it at the moment. We will revisit this class later in LT10. Let us do some preparation work first.

LT2. In this Lab task, you will learn how to add an external library (a library that does not come with JDK or sometimes Java EE SDK) to the "ED-JDBC" project

LT2.1. In the "`ED-JDBC`" project, select "Libraries"
LT2.2. Right click the mouse and select "Add Library…"
LT2.3. In the "Add Library" window, select "Java DB Driver"
LT2.4. Click the "Add Library" button

Note: Expand on the "Libraries", you will see that the jar files "derby.jar", "derbyclient.jar" and "derbynet.jar" under "Libraries" together with "JDK 1.8 (Default)" – the default JDK library. If you cannot see these new "jar" files, your "Java DB" has not been set up properly. This means that "NetBeans cannot find them from its default location of GlassFish. Note the default location of your GlassFish's installation. Ask your tutor for help.

LT3. In this Lab task, you will learn how to create a Java class called `MyDB` that has the required methods to set up the database table and data for your future use

LT3.1. In the Projects tag, right click on the package "ed.jdbc" and select "New" > "Java Class…"

LT3.2. In the "New Java Class" window, enter "MyDB" in the "Class Name" field (Make sure "ed.jdbc" is selected in the "Package" field)

LT3.3. Click "Finish"

Note: NetBeans will create the file "MyDB.java" and display it in the editor window.

LT4. In this Lab task, you will learn how to write the `getConnection()` method that establishes the connection to Java DB via JDBC API. This method is in the `MyDB` Java class.

LT4.1. Select the "ED-JDBC" project

LT4.2. In the editor window, the file "MyDB.java" should be opened by now. (If not, expand on the "Source Packages" tree so that you can see "MyDB.java". Then, double click on "MyDB.java" will open the file in the editor window.)

LT4.3. Copy and paste the following code segment in "MyDB.java" (In case, your copy and paste causes compilation error, it may be due to "some hidden characters", especially the "double quotation" marks, in the pdf file. Be careful when you do copy and paste.)

```
public static Connection getConnection() throws SQLException,
IOException {
    System.setProperty("jdbc.drivers",
"org.apache.derby.jdbc.ClientDriver");
    String url = "jdbc:derby://localhost/sun-appserv-
samples;create=true";
    String username = "APP";
    String password = "APP";
    return DriverManager.getConnection(url, username, password);
}
```

Note: The option "create=true" means that the database server will create the database "sun-appserv-samples" if it does not exist. In case the database exists, the connection will be established without affecting any existing data in the database.

LT4.4. In the editor window, right click the mouse and select "Format" will fix the formatting of the source code

LT4.5. In the editor window, right click the mouse and select "Fix Imports" to fix the import statements. Make sure you select "java.sql.Connection"

LT5. In this Lab task, you will learn how to write the `createMyuserTable()` method (in `MyDB` class) that creates the database table MYUSER using SQL statements via JDBC.

Note: The MYUSER table has the following fields:

UserId	6 characters, primary key
Name	30 characters
Password	6 characters
Email	30 characters
Phone	10 characters
Address	30 characters

SecQn	60 characters	(nominated Security Question)
SecAns	60 characters	(Answer to SecQn)

LT5.1. Copy and paste the following code segment into "`MyDB.java`"

```
public void createMyuserTable() {
     Connection cnnct = null;
     Statement stmnt = null;
     try {
         cnnct = getConnection();
         stmnt = cnnct.createStatement();
         stmnt.execute("CREATE TABLE MYUSER ( "
             + " UserId CHAR(6) CONSTRAINT PK_CUSTOMER PRIMARY KEY, "
             + " Name CHAR(30), Password CHAR(6), Email CHAR(30), "
             + " Phone CHAR(10), Address CHAR(60), "
             + " SecQn CHAR(60), SecAns CHAR(60))");
     } catch (SQLException ex) {
          while (ex != null) {
               ex.printStackTrace();
               ex = ex.getNextException();
          }
     } catch (IOException ex) {
          ex.printStackTrace();
     } finally {
          if (stmnt != null) {
               try {
                    stmnt.close();
               } catch (SQLException e) {
               }
          }
          if (cnnct!= null) {
               try {
                    cnnct.close();
               } catch (SQLException sqlEx) {
               }
          }
     }
}
```

LT5.2. Remember to format the code and fix the import statements. Make sure you select "`java.sql.Statement`"

LT6. In this Lab task, you will create the `dropMyuserTable()` method (in `MyDB` class) that deletes the database table MYUSER from the database using SQL statements via JDBC.

LT6.1. Copy and paste the following code segment into "`MyDB.java`"

```
public void dropMyuserTable() {
     Connection cnnct = null;
     Statement stmnt = null;
     try {
         cnnct = getConnection();
         stmnt = cnnct.createStatement();
         stmnt.execute("DROP TABLE MYUSER");
     } catch (SQLException ex) {
          while (ex != null) {
               ex.printStackTrace();
               ex = ex.getNextException();
```

```
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            if (stmnt != null) {
                try {
                    stmnt.close();
                } catch (SQLException e) {
                }
            }
            if (cnnct!= null) {
                try {
                    cnnct.close();
                } catch (SQLException sqlEx) {
                }
            }
        }
}
```

LT6.2. Remember to format the code and fix the import statements, if needed.

LT7. In this Lab task, you will create a Java class called `Myuser` to hold the data related to the records in the MYUSER table

LT7.1. Create the Java class Myuser in the "ed.jdbc" package [See LT3 for details]

LT7.2. Copy and paste the following code segment to "`Myuser.java`"

```
    private String userid;
    private String name;
    private String password;
    private String email;
    private String phone;
    private String address;
    private String secQn;
    private String secAns;
```

Note: In case you do not know what these statements mean, they actually declare the required instance variables for the Myuser class. These variables have a one-one correspondence with the fields in the MYUSER database table. By doing so, a Myuser object in the program will then correspond to a record in the database table.

LT7.3. Right click at a position near these instance variables just added (avoid clicking on the area within comments) and select "Insert Code…"

LT7.4. In the "Generate" prompt, select "Constructor…" to create a constructor of the `Myuser` class

LT7.5. In the "Generate Constructor" window, click "Select All" and click "Generate"

Note: NetBeans will then generate a constructor for you that accepts all pa

LT7.6. Repeat LT7.3

LT7.7. In the "Generate" prompt, select "Getter and Setter…" to create the getters and setters of the `Myuser` class

LT7.8. In the "Generate Getters and Setters" window, click "Select All" and click "Generate"

Note: NetBeans will then generate all getters and setters for you.

LT8. In this Lab task, you will create the `addRecords()` method in "`MyDB.java`". This method adds multiple Myuser records to the database table (MYUSER) using SQL statements via JDBC.

LT8.1. Select the "`MyDB.java`" tab

LT8.2. Copy and paste the following code segment to "`MyDB.java`"

```
public void addRecords(ArrayList<Myuser> myUsers) {
```

```
        Connection cnnct = null;
        PreparedStatement pStmnt = null;
        try {
            cnnct = getConnection();
            String preQueryStatement
                = "INSERT INTO MYUSER VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
            pStmnt = cnnct.prepareStatement(preQueryStatement);

            for (Myuser myuser : myUsers) {
                pStmnt.setString(1, myuser.getUserid());
                pStmnt.setString(2, myuser.getName());
                pStmnt.setString(3, myuser.getPassword());
                pStmnt.setString(4, myuser.getEmail());
                pStmnt.setString(5, myuser.getPhone());
                pStmnt.setString(6, myuser.getAddress());
                pStmnt.setString(7, myuser.getSecQn());
                pStmnt.setString(8, myuser.getSecAns());

                int rowCount = pStmnt.executeUpdate();
                if (rowCount == 0) {
                    throw new SQLException("Cannot insert records!");
                }
            }
        } catch (SQLException ex) {
            while (ex != null) {
                ex.printStackTrace();
                ex = ex.getNextException();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            if (pStmnt != null) {
                try {
                    pStmnt.close();
                } catch (SQLException e) {
                }
            }
            if (cnnct!= null) {
                try {
                    cnnct.close();
                } catch (SQLException sqlEx) {
                }
            }
        }
}
```

LT8.3. Remember to format the code and fix the import statements. Make sure you select "`java.sql.PreparedStatement`" and "`java.util.ArrayList`"

Note 1: In this method, we use `PreparedStatement` object to define a dynamic query. Please note the differences between this and the one we used in `createMyuserTable()` method in LT5, which is a static query.

Note 2: In fact, you may want to use the `Statement` object to define a static query, like the one we used in `createMyuserTable()` method, to insert data into your database table. In that case, the code will then be similar to `createMyUserTable()`. The only difference is the SQL statement being used.

LT9. In this Lab task, you will learn how to establish a database connection in NetBeans for viewing the data in Java_DB.

LT9.1. Select the "Services" tab in NetBeans

LT9.2. Expand on "Databases"

LT9.3. Right click on the connection "jdbc:derby://localhost:1527/sun-appserv-samples [APP on APP]"

LT9.4. Select "Connect..."

Note: NetBeans will then establish the connection. In case the database server has not been started, NetBeans will start the database server before establishing the connection. Once connected, note that the icon has been changed.

LT9.5. Expand on this "sun-appserv-samples" connection > "APP" > "Tables"

Note: You will see the database tables for user "APP" – there is none at the moment. We will create one soon via a JDBC program.

LT10. In this Lab task, you will learn how to write a JDBC program that utilizes the methods that you did in the past several LTs to create the actual MYUSER database table and load 5 Myuser records into the table

LT10.1. Select the "`SetUpMyUser.java`" tab in the editor window

LT10.2. Copy and paste the following code segment in the "`main`" method

```
MyDB mydb = new MyDB();

/*
 * drop table first for a clean start
 * may cause error if table does not exist
 */
mydb.dropMyuserTable();
mydb.createMyuserTable();

ArrayList<Myuser> aList = prepareMyuserData();
mydb.addRecords(aList);
```

Remember to format the code and fix the imports. Make sure you select "`java.util.ArrayList`"

Note: The program cannot be compiled at the moment because the method "`prepareMyuserData()`" does not exist. You will add this method next.

LT10.3. Copy and paste the following code segment in the class but outside the "`main`" method

```
public static ArrayList<Myuser> prepareMyuserData() {
    ArrayList<Myuser> myList = new ArrayList<Myuser>();

    Myuser myuser1 = new Myuser("000001", "Peter Smith", "123456",
              "psmith@swin.edu.au", "9876543210", "Swinburne EN510f",
              "What is my name?", "Peter");
    Myuser myuser2 = new Myuser("000002", "James T. Kirk", "234567",
              "jkirk@swin.edu.au", "8765432109", "Swinburne EN511a",
              "What is my name?", "James");
    Myuser myuser3 = new Myuser("000003", "Sheldon Cooper", "345678",
              "scooper@swin.edu.au", "7654321098", "Swinburne EN512a",
              "What is my last name?", "Cooper");
    Myuser myuser4 = new Myuser("000004", "Clark Kent", "456789",
              "ckent@swin.edu.au", "6543210987", "Swinburne EN513a",
```



```
                "What is my last name?", "Kent");
    Myuser myuser5 = new Myuser("000005", "Harry Potter", "567890",
                "hpotter@swin.edu.au", "6543210987", "Swinburne EN514a",
                "What is my last name?", "Potter");

    myList.add(myuser1);
    myList.add(myuser2);
    myList.add(myuser3);
    myList.add(myuser4);
    myList.add(myuser5);

    return myList;
}
```

Remember to format the code and fix the imports, if necessary.

LT10.4. Run the program "`SetUpMyUser.java`" (Please make sure that the database server has been started before running this program)

Note: If "MYUSER" table does not exist in the database server, NetBeans will display an error message saying that "MYUSER" table does not exist. This is fine because we call the "`dropMyuserTable()`" first and then the "`createMyuserTable()`". If you run this again, it should work just fine.

LT10.5. Go to the "sun-appserv-samples" connection, expand on "APP", right click on "Tables" and select "Refresh"

Note: You will then see the MYUSER table. This means the MYUSER table has been created.

LT10.6. Right click on "MYUSER" table and select "View Data…"

Note: You will then see the 5 Myuser records has now been loaded to the actual database table.

Final Word

Now, you have created a JDBC program "SetUpMyUser.java" under the lab instructions. Good luck with your NetBeans experiences and "simple" Java programming.

Next week, we will be doing simple JPA application that runs on normal JVM. This also gives you some idea about how Hibernate – the first (?) Java ORM framework – works.