Software

To finish the 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 [JavaDB is the database server that comes with GlassFish.]

Aim

Write a program that makes use of Java Persistence API to add a new record to a database table. We do this as JPA application. My intention is for you to have some exposures on programming JPA without using Java EE. The techniques of programming JPA applies to other ORM technologies such as Hibernate.

Figure 1 shows a rough design of "MyuserApp" and its related classes. It also shows related the roles of these classes in the project. The application "MyuserApp.java" will handle the input from users to perform the CRUD operations of MYUSER database table via the Myuser class using JPA (or, ORM). In order to manage to search relevant information from the MYUSER database (without executing any SQL statements) and return with proper Myuser objects, it is better to have a separate JPA controller class rather than doing everything in the Myuser class. We call this JPA controller class "MyuserDB", I name it as "…DB" because it emulates the required database functions.

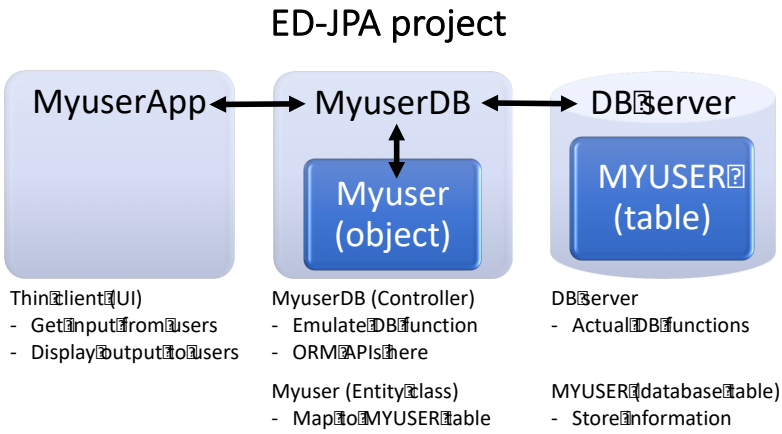

Figure 1 A rough design of MyuserApp

PreLab Task

Before this Lab, you need to first create the database table MYUSER in "sun-appserv-samples" on "Java DB" and load some data into the table, as documented in Lab_02_Database_Connectivity. If you have completed the program in Lab_02_Database_Connectivity, you are all set except you manually delete the table or alter the data in between. In that case, you can rerun the program (assuming you have not changed anything) to recreate the database table MYUSER and reload some data in the table.

Overview of Lab Tasks

In this Lab, you will learn to perform the following tasks in programming an entity class:

LT1. Create a Java Class Library project
LT2. Program an Entity Class using the Java Persistence API (JPA)
LT3. Create a Java Application Project
LT4. Add a project to another project
LT5. Program a JPA controller class in the application project
LT6. Create a DTO for the entity class created in LT2 above
LT7. Program the thin client for the application project
LT8. Program the remaining methods required
LT9. Run the application and verify the results

Lab Tasks

This Lab should be run on MS Windows Platform

LT1. In this Lab task, you will create a new Java Class Library project in NetBeans called "ED-Entity"

LT1.1 Select "File" > "New Project"
LT1.2 Choose "Java with Ant" > "Java Class Library", then click "Next"
LT1.3 Enter "ED-Entity" in the Project Name field
LT1.4 Click "Finish"
Note: NetBeans creates the project. There is no Main class as suggested by the type of the project that we choose.

LT2. In this Lab task, you will learn how to program the "Entity Class", Myuser. Make sure that you have completed the PreLab Task above.

Note: We cannot reuse the "Myuser.java" class created in previous Lab because it is a normal Java class. You need to follow the instructions below to perform this task.
LT2.1 Select "ED-Entity " project
LT2.2 Right click the mouse and select "New" > "Entity Class from Database…" (See Note below if you cannot find this option)

Note: If you cannot find "Entity Class from Database…", select "Other…". Then, in the "New File" window, choose "Persistence" in the "Categories:" selection box, then choose "Entity Classes from Database" in the "File Types:" selection box. Then, click "Next".

LT2.3 In the "New Entity Classes from Database" window, select "jdbc:derby://localhost:1527/sun-appserv-samples [APP on APP]" (the one used in your PreLab Task) in the Database Connection field and put the "MYUSER" table in the "Selected Tables" text are (Make sure the "Include Related Tables" check box is checked)

Note: If you cannot find the "MYUSER" table in the "Available Tables:" field, make sure your database connection string is right and you have set up the database properly with the proper username "APP" and password "APP". You may want to remove the database and recreate one with the right settings.

LT2.4 Click "Next"
Note: A new dialog box appears with "MYUSER" in the "Class Names" field and "ED-Entity" in the Project field
LT2.5 Enter "entity" in the Package field
LT2.6 Make sure that the following check boxes are checked
a. "Generate Named Query Annotations for Persistent Fields"

Note: By checking the box, NetBeans will create some standard named queries like
"Myuser.findAll" to find all myusers in the database table
"Myuser.findByUserid" to find a particular myuser using userid (userid is the primary key of the table)
… (others omitted) …

b. "Generate JAXB Annotations"
Note: By checking the box, NetBeans will create some standard annotations that allow the entity class to be used via JAXB web services. We do not need this actually.

c. "Create Persistence Unit"

LT2.7 Click "Finish"

Note: NetBeans will automatically generate the entity POJO "Myuser.java" for you, including some named queries.

Note: If you do not have any methods to update the data, this entity class is now complete.

Note: This entity class is a Data Access Object (DAO) that is responsible for holding the required information for the records in the MYUSER database table.

Note: In the source folder, you can see a "package" called "META-INF". The file "persistence.xml" is the persistence unit of Myuser, in xml format and act as a config file.

Note: In the Libraries folder, there are three additional EclipseLink libraries. They are the standard JPA libraries.

Question to think further / deeper: Why "Myuser.java" needs to implement the "Serializable" interface?

LT3. Create a new Java Application Project in NetBeans called "ED-JPA"

LT3.1 Select "File" > "New Project"

LT3.2 Choose "Java with Ant" > "Java Application", then click "Next"

LT3.3 Enter "ED-JPA" in the Project Name field

LT3.4 Check the "Create Main class" check box and enter "ed.jpa.MyuserApp" in the corresponding text field

LT3.5 Click "Finish"

Note: NetBeans now creates the project for you. You will also have a "public static void main()" method in MyuserApp.java.

In the following, we are going to add the "ED-Entity" project to the "ED-JPA" project. The actual project jar file will be added. By doing so, we achieve reusability.

LT4. Add a project to another project

LT4.1 Right click on "Libraries" under the "ED-JPA" project, then select "Add Project…"

LT4.2 In the "Add Project" windows, select "ED-Entity"

LT4.3 Click "Add Project JAR Files"

Note: NetBeans now adds the "ED-Entity.jar" file to the Libraries.

LT5. Program a JPA controller class in the ED-JPA project

LT5.1 In the ED-JPA project, right click on package "ed.jpa" under "Sources Packages"

LT5.2 Select "New", then "Java Class…"

LT5.3 In the "New Java Class" window, enter "MyuserDB" in the "Class Name" field

LT5.4 Click "Finish"

LT5.5 Copy and paste the following code segment into MyuserDB

```
private EntityManager em = null;

public MyuserDB() {
    // using default persistence unit defined in the persistence.xml file
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("ED-EntityPU");
    em = emf.createEntityManager();
}

public EntityManager getEntityManger() {
    return em;
}
```

Note: This time, "Fix imports" does not work because the JPA libraries are not in the standard Java libraries. We will add them to the project in the next learning task.

LT5.6 In this learning task, we need to add the JPA libraries (and Java DB libraries) to the "ED-JPA" project

a. Right click on the "Libraries…" in the "ED-JPA" project and select "Add Library…"

b. In the "Add Library" window, select "EclipseLink (JPA 2.1)" and "Java DB Driver" together (holding the "Ctrl" key allows you to make multiple selections)

c. Click "Add Library"

Note: NetBeans will then add the relevant libraries into the project.

d. Remember to fix imports…

LT5.7 Copy and paste the following code segment after the `getEntityManager()` method

```
public Myuser findMyuser(String userid) {
    return em.find(Myuser.class, userid);
}

public boolean createMyuser(Myuser myuser) throws Exception {
    try {
        if (findMyuser(myuser.getUserid()) != null) {
            // myuser exists already
            return false;
        }
        // myuser does not exist in database
        em.getTransaction().begin();
        em.persist(myuser);     // to add an object to database
        em.getTransaction().commit();

        return true;
    } catch (Exception ex) {
        throw ex;
    }
}
```

Remember to fix imports and …

LT6. Program the DTO (data transfer object) class for "Myuser.java", we call this "MyuserDTO.java"

LT6.1 Create a new Java class called "MyuserDTO" in the package "ed.jpa"

LT6.2 Make sure that "MyuserDTO" implements the "java.io.Serializable" interface

Hint: **Add** "**implements Serializable**" in between "public class MyuserDTO" and "{". Remember to fix imports.

LT6.3 Copy and paste the following code segments into "MyuserDTO" class

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

public MyuserDTO(String userid, String name, String password,
        String email, String phone, String address,
        String secQn, String secAns) {
    this.userid = userid;
    this.name = name;
    this.password = password;
    this.email = email;
    this.phone = phone;
    this.address = address;
    this.secQn = secQn;
    this.secAns = secAns;
}
```

LT6.4 Generate the getters of these instance variables (Only getters are required. No setters.)

Question to think further / deeper: What is the role of a DTO? How is it different from a DAO?

LT7. Program the thin client, MyuserApp, to add data to database via JPA

Create a new Java class called "MyuserAPP.java" in the package "ed.jpa" – refer to create a new Java class in LT5 and LT6

LT7.1 Copy and paste the following code segment to "MyuserApp.java"

```
private MyuserDB mydb;

public MyuserApp() {
    mydb = new MyuserDB();
}

public static void main(String[] args) {
    MyuserApp client = new MyuserApp();

    // assuming inputs from keyboard or any GUI
    MyuserDTO myuserDTO = new MyuserDTO("000001", "Peter Smith", "123456",
            "psmith@swin.edu.au", "9876543210", "Swinburne EN510f",
            "What is my name?", "Peter");
    boolean result = client.createRecord(myuserDTO);
    client.showCreateResult(result, myuserDTO);

    // assuming inputs from keyboard or any GUI
    MyuserDTO myuserDTO2 = new MyuserDTO("000006", "David Lee", "654321",
            "dlee@swin.edu.au", "0123456789", "Swinburne EN510g",
            "What is my name?", "David");
    result = client.createRecord(myuserDTO2);
    client.showCreateResult(result, myuserDTO2);
}

public void showCreateResult(boolean result, MyuserDTO myuserDTO) {
    if (result) {
        System.out.println("Record with primary key " + myuserDTO.getUserid()
            + " has been created in the database table.");
    } else {
        System.out.println("Record with primary key " + myuserDTO.getUserid()
            + " could not be created in the database table!");
    }
}

public boolean createRecord(MyuserDTO myuserDTO) {
    return mydb.createRecord(myuserDTO);
}
```

Note: In the next task, we will program the `createRecord()` method in the MyuserDB class.

LT8. Program the remaining methods required in the MyuserDB class

LT8.1 Copy and paste the following code segment in the MyuserDB class

```
public boolean createRecord(MyuserDTO myuserDTO) {
    Myuser myuser = this.myDTO2DAO(myuserDTO);

    boolean result = false;
    try {
        result = this.createMyuser(myuser);
    } catch (Exception ex) {
    }

    return result;
}

private Myuser myDTO2DAO(MyuserDTO myDTO) {
    Myuser myuser = new Myuser();

    myuser.setUserid(myDTO.getUserid());
    myuser.setName(myDTO.getName());
    myuser.setPassword(myDTO.getPassword());
    myuser.setEmail(myDTO.getEmail());
    myuser.setPhone(myDTO.getPhone());
```

```
    myuser.setAddress(myDTO.getAddress());
    myuser.setSecqn(myDTO.getSecQn());
    myuser.setSecans(myDTO.getSecAns());

    return myuser;
}
```

LT8.2 Remember to save all the modifications made

LT9. Run "MyuserApp" and verify the results (or run "ED-JPA")

Note: has done: start "Java DB" server

Final Word

Start from next week onwards, we will do "Java EE" stuff – the real deal. That will be more **fun**, more **challenging** and, I can guarantee you, "MORE FRUSTRATING" – You are warned.