Java EE Lab Activity 2: Persistence Units

We will work on this Activity in the Lab period. Submit+Demonstrate your final results to your Lab Professor for 1+2 marks (3% of course grade)

See Brightspace for due date.

Purpose: Learn about JEE persistence units. The address book application that we study in this course uses a persistence unit called `address-bookPU`. In general, a persistence unit defines a set of Entity classes (`Contact.java` in this case) to be managed by `EntityManager` instances in an application, and it specifies a `jta-data-source` (database) where the instances of entities are stored. The directory whose `META-INF` directory contains persistence.xml is called the root of the persistence unit.

At the end of this lab, you should be able to complete the steps necessary to configure a persistence application to store its data with an arbitrary database engine running on an arbitrary IP address.

Demonstration Questions: Be prepared to answer the following questions during your demonstration:

1. What changes did you make in order to have the primary key field "`id`" automatically generated for a new contact whenever a new contact is saved?
2. What changes did you make so that the primary key field "`id`" would not be shown on the "create" or "edit" page of the application?
3. What is the name of the persistence unit for your new application?
4. How does the programmer specify the set of entities (just one entity class in our case) for the persistence unit?

Note on Passwords: In case you are prompted for the password for the `sun-appserv-samples` database, you should be able to leave the username and passwords blank. You would need this password (username: `APP`, password `APP`) if you set up a new data source with a different name for the `sun-appserv-samples` database, but we will not be doing that. Also, the admin console (port 4848) of the Glassfish server is password protected, but as long as you don't set a password, you should be able to log in without one. If prompted for a password for the Glassfish Console, try `admin/admin` (you shouldn't be prompted unless you set a username and password, and it's easiest for our development purposes if we just leave it unset – we would not leave it unset with a production server).

Explore Address-book persistence.xml

Find the `persistence.xml` file in the `address-book` application. This file specifies a persistence unit by listing a set of entity classes and a database connection. The entity classes are Java classes which implement the information that would be given by an entity relationship diagram (ERD). In the case of the `address-book` application there is just one entity with no relationships to other entities. The

database connection specifies where the system will create the database tables that correspond to the entities (one entity in our case).

Notice that when you open `persistence.xml` in Netbeans, there are several tabbed views for displaying/editing the file's contents: `Design`, `Source`, and `History`. In the `Design` view, you can easily see the following items:

`Name`: the name of the persistence unit

`Provider`: the implementation of the Java Persistence API (JPA) to be used (Eclipselink is the default, which we will use in this course, and Hibernate is another option)

`Data Source`: this represents the database where the application's content (Entity instances) should be stored. The `java:comp/DefaultDataSource` expression is a JNDI name (Java Naming and Directory Interface) that corresponds to the `sun-appserv-samples` database.

`Table Generation Strategy`: determines whether the entity table(s) will be dropped from the database and re-created, which is desirable if we are in a phase of development that involves working on our Entity class, changing its properties. Any time we change our Entity properties, the table and the database used to store those Entity instances will need to change. If our Entity is finalized, we can set the strategy to `none`, or to `create` which will create the table(s) if not there. `None` or `create` will leave pre-existing tables and their data as they are.

`Include Entity Classes`: this important section is where we specify the set of Entity classes for the persistence unit. We can list all the Entity classes, or alternatively we can check the checkbox for including all Entity classes in the project.

Writing a JSF Application

Using basically the same process you used in the previous lab, you can quickly write an application like the Address Book application, from scratch, using Netbeans. Here we do it again for practice, and we go further, fixing the new application to auto-generate the `id` field, the way the `address-book` does:

1. New **Maven** Web Application (File->New Project->Maven Project)
2. Create the `Contact` entity class. Normally when writing a new program, the programmer would not yet have database tables, and could create this java class (java source code) first from scratch as a starting point, but here we will use the same technique we used in the previous lab because we happen to have an existing table on which to base our entity. Assuming you have previously run the `address-book` application, which will create the `Contact` table, Netbeans can re-create a `Contact.java` entity class from the table (right click on Project, `New->Entity Classes from Database`, and select the `Contact` table). There should now be a `Contact.java` class in your project, in the package you specified for it.
3. Create the JSF pages to do CRUD operations on the Entity (right click on Project, `New->JSF Pages from Entity Class`).

4. Change the `Contact.java` class so that the `id` field of each new added contact is generated automatically, the way it is in the original `address-book`. Figure out how to do this by examining the `Contact.java` class in both the original `address-book` project and in your new project, looking especially at the annotations just above the `id` field (the java variable named `id`). Make the annotations above the `id` field the same in your new `Contact.java` class as they are in the `address-book` project.
5. Regarding automatically generated `id` fields, do the same comparison with the JSF pages (`Web Pages->Contact->Create.xhtml` and `Pages->Contact->Edit.xhtml` in both projects). You will need to delete the `id` field from those pages so that the user will not attempt to set or change an id manually. You can also re-arrange the ordering of the other fields to match the ordering on the create page of the original address book.

Make Change to Persistence Unit

The next steps deal with taking control of where the data for the new application will be stored. We will install a database program of our own (MariaDB, which is a version of MySQL), and configure the new application to use that database engine instead of the Derby database that comes with Glassfish.

6. Install MariaDB, a fork of MySQL, in order to migrate your application to use an external MariaDB database. (Oracle, MySQL, and Postgres are all options – MariaDB is a convenient example to use).
    a. https://downloads.mariadb.org/ (use the dropdowns to select your operating system and architecture)
    b. 10.6.5 is the current version as of this writing
    c. The 64-bit Windows-msi installer is known to work well
    d. Default settings in the installation wizard are fine, but set a password for the `root` user.
7. Sign-in to your MariaDB database engine with the MySQL Client program accessible from the Start menu in Windows (or use the HeidiSQL GUI if you prefer) and create a database to be used for your Persistence Unit. The SQL code to create a database named “mydatabase” is

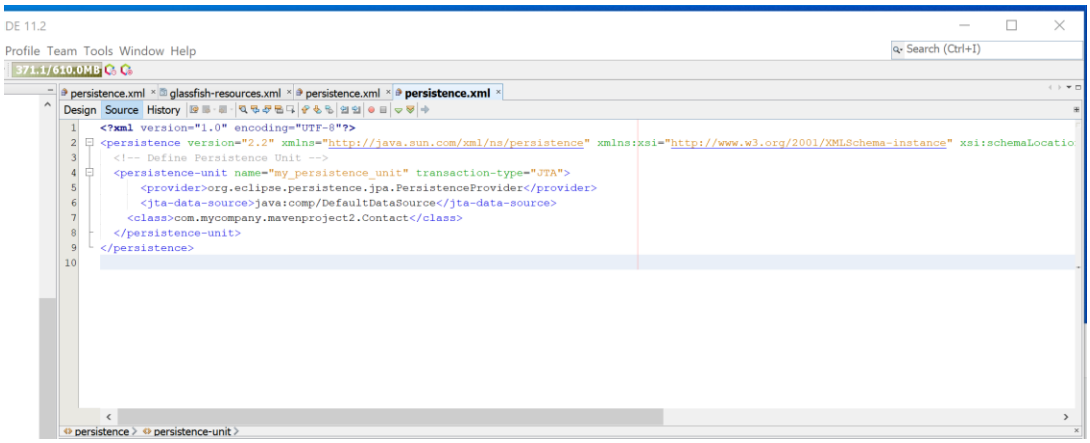
    ```
    >create database mydatabase;
    ```

    You should choose a more meaningful name than “mydatabase”, perhaps "cst8218".

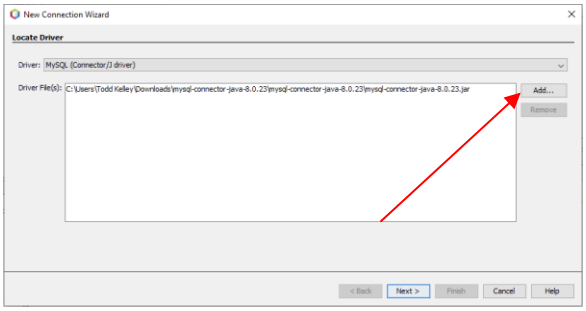
8. Install the Connector/J driver (do not use their recommended download at the link below, because it is for installing all of MySQL) to allow Java to interact with a MariaDB (MySQL) database. Download the driver from https://dev.mysql.com/downloads/connector/j/
    a. Notice there is a "no thanks, just start my download" option on the page that asks you to create an account.
    b. For Windows, do **not** choose the "Windows" and do **not** choose the "recommended windows download". Choose “**Platform Independent**” as the Operating System.
    c. The current version as of this writing is `mysql-connector-java-8.0.27`

d. Unzip the mysql connector driver to a convenient place (e.g. desktop or downloads), and we will browse to it from the Netbeans wizard later in Step 10.b.ii.

9. In your new Project, the clone of the address-book app, you'll need to modify the file, `persistence.xml` in order to switch the persistence unit to a different database engine (see screenshot below).
    a. Using the Source tab, compare your new file to the `address_book` version of `persistence.xml`
    b. Make the new persistence unit use the Java Trasaction API `JTA` transaction type (as in the address_book version). When you do this, you will see two certificate pop-up windows which are expected, and you can accept those.
    c. Use EclipseLink as the Java Persistence API `jpa` provider (as in the address_book version)
    d. Specify the `DefaultDataSource` JNDI name as the `jta-data-source` (as in the address_book version)



10. Now that we have updated the new persistence.xml to specify the `JTA` transaction type, in Design View of the new `persistence.xml` select `New Data Source…` and go through the wizard
    a. Pick a meaningful JNDI name for your new datasource (MariaDB?)
    b. For a Database Connection, choose `New Database Connection` and go through the wizard
        i. For the Driver, you will want to select Connector/J for MySQL (MariaDB)
        ii. For the Driver File, you will use the `Add` button to add the Connector/J jar at the location you unzipped it (Step 8c above)

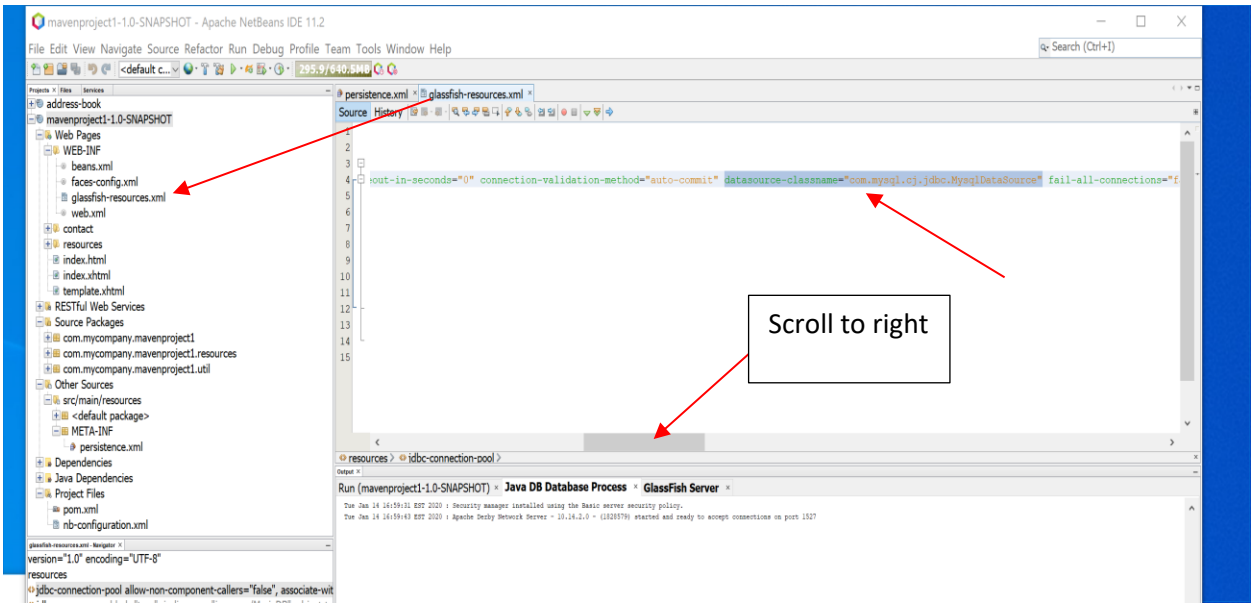


iii. On the Customize Connection Screen, you will build up the JDBC URL (the information you enter is stored in a file called `glassfish-resources.xml`) :

1. `Host`: the IP address of your MariaDB engine (often `localhost`)
2. `Database`: the name of the database you created in Step 7 above (Do not use `mysql` for your database because `mysql` is the name of the database used by the MySQL (MariaDB) database engine itself for its own purposes, like managing database users and their permissions!)
3. `Username`: the name of the database engine user created in Step 6: using the root user is sufficient for our purposes. (best practice is to create a non-root database user, and ensure that user has permissions on the database created in Step 7, and can connect from the machine running the Glassfish server (often `localhost`)
4. `Password`: the MariaDB password for your database user (select `remember password`)
5. You can select "`Finish`" from this screen

11. Notice the table generation strategy in your persistence.xml: set it to a setting that will create the table.

12. After creating the new DataSource, you will have a new file called `glassfish-resources.xml`. The DataSource classname is outdated in `glassfish-resources.xml`, since it changed to "`com.mysql.cj.jdbc.MysqlDataSource`" in the latest Connector/J, so we need to change that according to the screenshot below:



13. You should now be able to run your new application, and it should create the `CONTACT` table in your MariaDB (MySQL) database. Show your Lab Professor that you can add a contact to your MariaDB database. If you want your testing contacts data to stay in the database, you should verify the setting in persistence.xml so that the application does not "drop and create" the database tables every time it runs (which is convenient if you're working on the Entity itself, adding or removing fields, which we are not doing, yet).

Demonstration

Before your demonstration begins, run your new application that uses MariaDB as the database, and add a new contact. Be ready to show the contents of the contact table in the MariaDB database. From the MySQL client program you would issue the following commands (where `<mydatabasename>` is the name of the database you created in MariaDB)

```
use <mydatabasename>;

show tables;

select * from contact;
```

Be prepared to answer the demonstration questions at the top of this document.