**Building one JPA project from another JPA project**

Maven is the tool that we are using to marshal all the components together that our JPA application needs to run. The pom.xml file that I have given you tells Maven what to pull together and which version of those tools to properly configure the application.

The persistence.xml file configures JPA itself. Most of the parameter values in persistence.xml do not have to be modified at all from one project to the next, but others do.

In this little "cookbook", I want to show you a quick step by step process for using the CarClub files to create a new project.

1. Start with the CarClub project here.
    a. Download CarClub as a .zip file by hitting the green "Code" button.
    b. Move that file to wherever you will build your new project.
    c. Unzip the .zip file.
2. Open CarClub in IntelliJ as a **project**.
    a. Start IntelliJ.
    b. Use File | Open and navigate to the pom.xml file in CarClub.
    c. Tell IntelliJ that you want to open it as a project and to trust it as a project.
3. In the pom.xml file:
    a. From a Java perspective, it is in a package: csulb.cecs323.model. You can probably leave that alone for this purpose. If you choose to change that, line 65 has a <mainClass> parameter that you will need to change.
    b. The pom.xml file calls out the main program for your application. That is also called out in the <mainClass> parameter. You will probably want to change that from CarClub.
    c. The Java version called out in <java.version> will need to be updated. Just be sure that you have that version of the JDK installed. It's entirely possible that IntelliJ will prompt you to download the JDK if you do not have it installed, but it's best to be sure.
4. Back in IntelliJ, over to the left in the Project navigator, under src\main\java\csulb.cecs323\app will be CarClub. You need to rename that to match what you have for your main in the pom.xml.
    a. Right click it in IntelliJ | Refactor | Rename and give it a name more in keeping with your new project.
    b. That will also change the class name and any references that you make to that class in your main code.
5. In the persistence.xml file:
    a. Change the persistence unit name:

i. The persistence-unit name (line 5) will be “CarClub”. Change that to something more in keeping with your new project.

ii. Go into your main, and find:

```
EntityManagerFactory factory =
Persistence.createEntityManagerFactory("CarClub");
```

iii. Change “CarClub” to match the persistence-unit name that you put into your persistence.xml file.

b. Change the database name:

i. Find the property named: “javax.persistence.jdbc.url”. Right now, that tells JPA to tell Derby to build your database at /database/CarClub under the root of your project.

ii. Change CarClub to something more in keeping with your new project.

iii. You can move the database to anywhere that you want, but personally, I find that having it here, right under the root of the project is handy. So far, each JPA project that I’ve done always has its own little dedicated Derby database. In general, you will share a database with many other applications, but this is simpler, and simpler is always a good thing.

c. Choose your EclipseLink logging level:

i. Find the property “eclipselink.logging.level”.

ii. You can find a list of the logging levels that you can use to filter which of those EL log messages that you see here.

iii. The value “OFF” will turn off all EclipseLink logging.

6. Changes to the Java code

a. Changing the Logging through the Logger class:

i. In your main, you can set the logging level that the Java Logger instance pointed to by LOGGER uses. The variable LOGGER is a static variable in your main class (whatever you chose to name it). Anywhere in main, you can alter the logging level.

ii. To turn logging off altogether: LOGGER.setLevel(Level.OFF);

iii. Other logging levels can be found here.

iv. The nice thing about using the Java Logger is that you do not have to add/remove System.out.println calls when you want debugging or not, you just change the logging level.

b. Delete the classes that you have under \src\main\java\csulb.cec323\model. You will add new ones there for your project.

7. Changes to the seed-data.sql file

    a. seed-data is a handy way to load your tables up with sample data, particularly if you are still having JPA drop and recreate the tables each time that you run the application. JPA runs that for you once the tables have been created.

    b. The location of seed-data.sql and its name is in persistence.xml as the value for “javax.persistence.sql-load-script-source”.

    c. You will want to remove all the records from seed-data.sql and replace them with inserts into your new tables as you develop your project. Otherwise, you will receive a lot of error messages when the database tries to update tables that do not exist.