Workshop 4: Java Persistence API and the EJB component model

DAT076/DIT126 Chalmers/Göteborgs Universitet
Joachim von Hacht

Objectives

The goal is to replace the usage of the in-memory shop-model with a real relational database. This will force us to do some modifications of the model and replace the “in-memory”-code with real database code using JPA and EJB. The final outcome should be Arquillian tests covering all methods in the ProductCatalogue. We will use:

- GlassFish and a relational database, JavaDB (aka Derby).
- Java Persistence API (JPA)
- Enterprise JavaBeans (EJBs) to host the JPA code.
- Arquillian (embedded container) to test the persistence layer.

Please, have a look at the code samples, everything you need should be there. Also links in slides for more detailed information.

The a sample database

Start with a quick look at a sample database.

1. In NetBeans go to Services tab > Databases > Right click “jdbc:derby.../sample” > Connect (possible, login/passwd = app/app .. or other ...).
2. Expand the node “jdbc”.... > APP > Tables > CUSTOMER.
3. Mark CUSTOMER > Right click > View data. Change the SQL to the below and click run (button in toolbar with small green triangle (leftmost));
   ```
   select * from APP.CUSTOMER where customer_id < 150;
   ```
4. It's also possible to add, delete or update records in the table. Use icons in the toolbar (in the listing window) or cursor over the table data > Right click > .... or just click in a cell and alter the value. Click commit-button in toolbar (icon with table and small check mark) to commit the change.

Creating a database

First we'll create a database. This is an application external operation.

1. In NetBeans go to Services tab > Databases > Java DB (right click) > Create Database…
2. Fill in (and then OK):
    a. Database Name: shop
    b. User Name and Password: app
3. A new connection should show up.
4. Connect (right click) and inspect the (empty) database.

Deleting a database

You possible have need for this. Test now or remember for future use.

1. Mark jdbc:derby://.../shop > right click, Delete.

If in trouble, stop database server, go to .netbeans-derby directory and delete the folder named as the database. Restart database server.

Implementation

The overall task is to rework the given shop model so that it can be mapped to a relational database.

Mapping the Shop model

1. If database deleted, recreate.
2. Download skeleton code. Open in NetBeans.
3. Identify the persistence model and add annotations for relevant classes in core package.
4. Try to generate and inspect created tables. Add a servlet as in the samples to trigger table creation (see sample jpa_map_single > DummyServlet).
   NOTE: Don’t write tests in servlet, just use to generate the tables!
   If result is unsatisfactory delete tables and retry (from inside NetBeans)

Managing the Shop model

Implement one method at time!

1. Implement some method in the AbstractDAO and let ProductCatalogue inherit.
2. Test using Arquillian
3. Repeat until all operations in IDAO interface (for ProductCatalogut) are implemented and tested (some methods need JPQL queries, see below)

NOTE: Arquillian can be tricky, really have to check everything!

1. Inspect files in src/test/resources (Arquillian config files).
2. Use the supplied Arquillian tests as a start. Comment out testProductGetByName.
3. If create()-method (in AbstractDAO) implemented it should be possible to run testPersistAProduct (also; all mappings must be correct). Try! If success inspect generated tables and data.

Queries with JPQL

It's possible to run JPQL from within NetBeans (no ';' - char last in queries, I'll think...)

- Mark the persistence.xml > Right click > Run JPQL. Enter query and click Run icon.

NOTE: Beware of types! JPQL queries must be type correct.

Have Fun … (Optional)

1. Now it's possible for any front-end to use the modified shop model as a back end. Replace the dependency on shop with jpa_shop_skel in pom.xml. Give it a try (probably need some tweaking).
2. Try to implement the JEE security model with a database backend