University of Central Lancashire
UCLan

CO3409 Distributed Enterprise Systems

Lab : Maven

Summary

Use NetBeans to develop an Enterprise Application and test suite using Maven to manage dependencies. The skills learnt here would be important in the development and testing of a full application.

Purpose

- On completion of this you should be able to
  - Create and use session beans and entity classes.
  - Develop test cases.
  - Use Maven

Things to Remember

Some things to remember...

- Run Netbeans as "Administrator".
- When running your application, if you are using a database please ensure your database is running first (see previous lab worksheets).
- To deploy an Enterprise Application, I suggest using Clean and Build and then Deploy.
- You will use Payara rather than Glassfish,
- To deploy an Enterprise Application, I suggest using Clean and Build and then Deploy.

Activities

1. Developing and Testing an Enterprise Application

Do the exercise in https://netbeans.org/kb/docs/javaee/maven-entapp-testing.html

> Note NetBeans has moved to Apache and the latest location of this exercise is https://netbeans.apache.org/kb/docs/javaee/maven-entapp-testing.html, but it is word for word the same exercise and they use *something* to represent **something**, which can be confusing. For other exercises, they've not copied the images across!).

Things to look out for!

- Use the Payara server rather than Glassfish when creating the Web Application.

- When you expand the freshly-created project, the javaee-web-api will be version 7, not 6.

- When creating the New Entity, it will be version 2.1 of EclipseLink.

- When fixing the imports for the test cases, use `org.junit.Assert`, rather than `junit.framework.Assert`, which has been deprecated by the name police for not fitting with some convention or other.

- In `testVerify()`, replace the declaration of `EJBContainer` with the following:

```
Properties props = new Properties();
props.put("installation.root",
   "C:\\Users\\dgcampbell\\Documents\\payara\\glassfish");
props.put(
     "org.glassfish.ejb.embedded.glassfish.configuration.file",
 "C:/Users/dgcampbell/Documents/payara/glassfish/domains/domain1/config/domain.xml"
        );
EJBContainer container =
   javax.ejb.embeddable.EJBContainer.createEJBContainer(props);
```

> **Remember** : To change the directory! Of course you're not going to have access to the `dgcampbell` area – amend appropriately.

- Do not change the properties to include `<glassfish.embedded-static-shell.jar>\*<INSTALLATION_PATH>\*/glassfish-4.0/glassfish/lib/embedded/glassfish-embedded-static-shell.jar</glassfish.embedded-static-shell.jar>`. This would use a local version of glassfish if you have glassfish installed. Glassfish doesn't come with the latest version of NetBeans, so we will let Maven download the jar.

- Instead of searching for “embedded-static-shell”, search for “**payara**” and select payara-embedded-all from fish.payara.extras –version 4.1.153 worked for me, whereas Version 4.1.2.174 didn’t. Version 5 doesn’t work. Make sure you change the **scope** to test. This component is a local application server, which you want to have to allow unit testing (without deploying the application to the real server) but you don’t want incorporated into your application when you deploy it for real because it will cause clashes.

- Don’t change the dependency for the glassfish jar to use the local `glassfish.embedded-static-shell.jar` property.

- When starting the server, if you want the database server to start at the same time, switch to the Services tab (If the Services tab is not available next to the Projects tab, use the Window menu item.) expand the Servers branch, right click on Payara Server and check the Start Registered Derby Server.

- When you run the test, it should fail. Fix the test.

2. Accessing the Session Bean from a Servlet

1. Create a new Servlet, Runner, in the package webTier.

2. Right-click in the body of the Servlet and choose **Insert Code**, **Call Enterprise Bean**, select the bean created above.

3. Notice that a private variable, `myEntityFacade`, has been created. Use it by inserting the following code into the Servlet:

```
myEntityFacade.insert(2);
int verify = myEntityFacade.verify();
out.println("verify: " + verify + "</p>");
```

4. Add the following link to `index.html` in the Web Pages:

```
<a href="http://localhost:8080/mavenwebtestapp/Runner">Run test
Servlet</a>
```

5. Run the application. The index page should be displayed. Click through to the servlet. Check the Payara log to see that items have been inserted and extracted.

3. Question Time

Note: I wouldn't be surprised if you needed to use the Internet to answer **[!]** questions. The rest should succumb to a mixture of common sense and shrewd guesswork.

1. What happens to Maven dependencies?

2. When is the junit jar used?

3. Where do the eclipse persistence jars come from? How do you know?

4. What happens if you delete the version of a dependency down to the first digit, retype the "."? What happens if you then delete the whole version?

5. Why do you not always use the latest version?

6. What does the packaging element do?

7. What is a unit test?

8. How does the JUnit runner know which of the methods in the test class are tests?

9. In the current test class, when will the following methods be called: `setUpClass`, `tearDownClass()`, `setup()`, `teardown()`? How do you know?

10. We had to make changes to `testVerify()` to make it work. What would you do to make all the other tests work?

    > Hint: it's not by copying the code from `testVerify()` into each test function.

11. What do `assertEquals()` and `fail()` do?

12. Can you think of a situation when `fail()` would be useful as more than a reminder that you've not thought about a generated test?

    > Hint: think about testing a situation where an exception should be thrown if you try to remove an item from empty collection.

13. **[!]** The runner class contains a private variable, `myEntityFacade`. How does that variable get initialised?

    > Hint: dependency injection

14. What does public abstract class `AbstractFacade<T>` mean?

15. Explain the operation of the following, paying particular attention to where the `EntityManager` comes from:

    ```
    public void create(T entity) {
    getEntityManager().persist(entity);
    }
    ```

16. **[!]** Why is the call to merge() needed in the remove method.

    > Hints: JPA, managed object

17. **[!]** What sort of Enterprise Java Bean is `MyEntityFacade`

    > Hint: look at the annotation

18. Where do you think jdbc/sample is mapped to an actual database?

    > Hint: look at `testVerify()`.

19. Look at `persistence.xml` in Other Sources. Examine both design and source modes. What does “drop-and-create” mean?

20. **[!]** What does ORM mean?

21. How does the eclipselink ORM identify the classes that need to be stored in the relational database?

    > Hint: how does it know that `MyEntity` is an Entity?

4. Further Reading

Introduction to Apache Maven | A build automation tool for Java projects

- https://www.geeksforgeeks.org/introduction-apache-maven-build-automation-tool-java-projects/

Introducing Maven (Video)

- https://www.lynda.com/Maven-tutorials/Introducing-Maven/794129-2.html