Exercises for Chapter 10

Exercise 10.1: A Bean Managed Persistence Entity EJB

This exercise demonstrates a simple BMP based *Ship* EJB. You will be creating a new Application archive for this exercise. You are also provided with a pre-built *EAR* file for convenience.

Building the application for Exercise 10.1

1. Download the *ex07-1.jar* file from the download site. Downloads may be available for the complete bundle of Exercises.
2. Copy the downloaded file and extract the file using *WinZip* or *jar* with `xvf` flags to *$EXAMPLES_HOME*. Example *c:\> EJBBook* directory.
3. Change directory to *$EXAMPLES_HOME\src\ex10_1*, referred to as *$EX10_1_HOME*.
4. Open a command prompt and type `Ant` or `Ant -buildfile build.xml`

This should create *$EXAMPLES_HOME\build\ex10_1* directory and compile the required files.

$EXAMPLES_HOME\pre-built\ex10_1 directory contains pre-packaged *.ear* file and the client *.jar* file for this exercise.

SQL for creating the Database table Ship is provided in *$EX10_1_HOME/sql* directory.

Database schema for the Cabin EJB

The build file has a target `create-ship-table` that can be used to create the database table. To use this target you need to have the Cloudscape server running. You can also use *cloudview* to open the database and create the required database table.

If you already have a Ship table, use the *cloudview* or command line utility to alter the table. You can also drop an existing table and create one with the following schema:

```
CREATE TABLE "Ship"
   (
      ID INT NOT NULL constraint pk_ship primary key,
      NAME CHAR(50),
      CAPACITY INT,
      TONNAGE DOUBLE PRECISION
   )
```

Building the Application Archive for Ex10_1

1. Run the Deploytool by typing in at the command prompt `c:\>deploytool`
2. Create a new application and call it *Titan10_1App.ear*.

A pre-built application is available in *$EXAMPLES_HOME/pre-built* directory.

3. Create a new EJB and give it a name **ShipEJB**. Add the EJB files in the `com.titan.ship` package to the EJB *.jar*.
4. Mark the EJB for **Bean Managed Persistence** and use `java.lang.Integer` as the Primary Key class.
5. Take the default selection for transaction management, i.e. container managed transaction. Select the default **Use Caller ID** for security Identity.
6. Click on the **Finish** Button to create the ShipEJB jar.
7. Select **File→Add to Application→ApplicationClient Jar** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex10_1/client_101.jar* and add it to the application. You can also create the application client **File→New→ApplicationClient** option.
8. Select **File→New→Web Component** from the menu to create a new web archive. Select all JSPs from *$EXAMPLES_HOME/build/ex10_1/jsp* directory. Go through the wizard setting up the EJB reference for the ShipEJB and giving a web context for the application.

This completes the configuration of the Titan10_1App.

Run the verifier to make sure there are no errors.

- ❖ Sometimes the verifier complains about transaction demarcation errors. It seems that the `xml` descriptors don't get updated. Simply go to the **Transactions** tab of the EJB and set the transaction attribute to **required** even though it shows **required**. This triggers the update of the descriptor files needed.

Deploy the application and mark the return of the client Jar. This should generate the required container classes and create the required **CustomerBeanTable** in the database.

Examine the ShipEJB characteristics.

This exercise consists of one client application. We will examine the EJB contents before we run the client application. This exercise demonstrates the characteristics of an Entity Bean with Bean Managed Persistence. In the previous chapters you have seen the Container Managed Persistence behavior. With EJB 2.0 CMP model the Bean class was defined as abstract class and the accessors and mutators were also abstract. The container call back methods were left empty. Vendor specific container tools generated the corresponding concrete classes and provided implementation of these methods.

In Entity Beans with BMP it is the developer's responsibility to provide all the information about the Bean's persistence mapping. In this exercise you can notice the following characteristics of the EJB:

- ♦ All the getters and setters for the Bean class have been defined.
- ♦ The container callback methods `ejbLoad, ejbCreate, ejbStore, and ejbRemove` are implemented.

- The BMP version also implemented `getEntityContext` and `setEntityContext.`
- The EJB also implements the corresponding finder methods that are declared in the Home Interface.

Examine the Standard EJB Descriptor File

```
<enterprise-beans>
        <entity>
              <display-name>ShipEJB</display-name>
              <ejb-name>ShipEJB</ejb-name>
              <home>com.titan.ship.ShipHomeRemote</home>
              <remote>com.titan.ship.ShipRemote</remote>
              <ejb-class>com.titan.ship.ShipBean</ejb-class>
              <persistence-type>Bean</persistence-type>
              <prim-key-class>java.lang.Integer</prim-key-class>
              <reentrant>False</reentrant>
              <security-identity>
                    <use-caller-identity/>
              </security-identity>
        </entity>
</enterprise-beans>
```

The `<persistence-type>` element is set to **Bean** instead of **Container**. This informs the *Deploytool* to generate the corresponding wrapper classes accordingly. Since this is not an EJB 2.0 CMP bean there are no `<cmp-field>` elements. The Bean code uses an implicit call using the JDBC DataSource object to get a connection. So a reference to the DataSource is not made explicitly in the descriptor.

The assembly descriptor of the *ejb-jar.xml* defines the transaction behavior and the security.

```
<assembly-descriptor>
        <method-permission>
              <unchecked/>
              <method>
                    <ejb-name>ShipEJB</ejb-name>
                    <method-intf>Remote</method-intf>
                    <method-name>getPrimaryKey</method-name>
                    <method-params/>
              </method>
              … … … Declarations for all the methods.
        </method-permission>
        <container-transaction>
              <method>
                    <ejb-name>ShipBean</ejb-name>
                    <method-intf>Home</method-intf>
                    <method-name>remove</method-name>
                    <method-params>
```

```
                        <method-param>
                              java.lang.Object
                        </method-param>
                  </method-params>
            </method>
            <trans-attribute>Required</trans-attribute>
      </container-transaction>
      … … … Declarations for all the methods with container
      transactions
</assembly-descriptor>
```

J2EE SDK provides the security unchecked empty element to allow for all users' access. Instead of creating an Everyone Role and assigning every method to them we take this approach of not performing a security check in essence with **<unchecked/>** element.

All methods are transactional and the `Required` value for the `<trans-attribute>` element ensures that the method is executed in the context of a transaction by creating a transactional context if needed or by using an active transaction context.

Examine and Run the Client Application

The client example provided as part of this exercise performs the following actions against the *ShipEJB*:

- Obtains a reference to the *Ship* EJB Home.
- Creates a *ShipEJB*.
- Updates the *ShipEJB* by calling some mutators.
- Uses the Finder to retrieve the EJB and reads the information from the EJB.
- Removes the EJB from the container, which physically deletes the persistence information from the database.

The following output demonstrates the client-side execution. Contrast it with the server-side log information to see the container callbacks and the distributed calls to the EJB for each client call.

```
C:\EJBBOOK\build\deploymentUnits\ears>runclient -client
Titan10_1App.ear -name Client_101
Initiating login ...
Username = null
Binding name:`java:comp/env/ejb/ShipHomeRemote`
Creating Ship 101..
Finding Ship 101 again..
Edmund Fitzgerald
50000.0
300
```

```
Removing Ship 101..
Unbinding name:`java:comp/env/ejb/ShipHomeRemote`
```

The server-side log information is shown below:

```
ejbCreate() pk=101 name=Edmund Fitzgerald
Obtained Connection: Trying to create statement
ejbStore() pk=101

ejbLoad() pk=101
setTonnage()
ejbStore() pk=101

ejbLoad() pk=101
setCapacity()
ejbStore() pk=101

ejbFindByPrimaryKey() primaryKey=101

ejbLoad() pk=101
getName()
ejbStore() pk=101

ejbLoad() pk=101
getTonnage()
ejbStore() pk=101

ejbLoad() pk=101
getCapacity()
ejbStore() pk=101

ejbLoad() pk=101
ejbRemove() pk=101
```

You should notice that on the first-time creation of the EJB the `ejbCreate` and `ejbStore` are called. Subsequently on every business method execution the container calls `ejbLoad` and `ejbStore`. This demonstrates that a design of an Entity Bean in this fashion will be quite expensive for the round trips. It is not recommended to code Entity Beans in this fashion, but is provided in the exercise to only demonstrate the concepts. Either by using Dependant Value objects or by way of bulk accessors the Entity Bean is supposed to be accessed to avoid multiple distributed calls and transactional over-head.

Finally, we remove the Entity Bean by calling remove on the EJBObject that calls the `ejbRemove` on the *ShipEJB* and deletes the record from the database.

Examine and run the Web Client for Client_61 program

As explained in the Building the Application Archive for Ex10_1 section, the web-archive is also deployed as part of the Application archive. The context of the web application is set to **titan10**.

Open the browser and go to the URI *http://localhost:8000/titan10/Client_101.jsp*

The *Client_101.jsp* performs the same operations as the *Client_101.java* program.