EJB Lab 5: Container-Managed Persistence Entity Bean

This lab has two parts. The application you will build will let a fleet administrator track the gas purchases of company drivers. In the first part, you will create a CMP entity bean that represents a Purchase -- each purchase's data will be saved persistently in a database. You will write a simple standalone client to test the entity bean.

In the second part, you will write a stateless session bean that provides higher-level access to the Purchase entity beans. You will then write a new client that takes advantage of the additional functionality.

Here is a partial UML diagram:

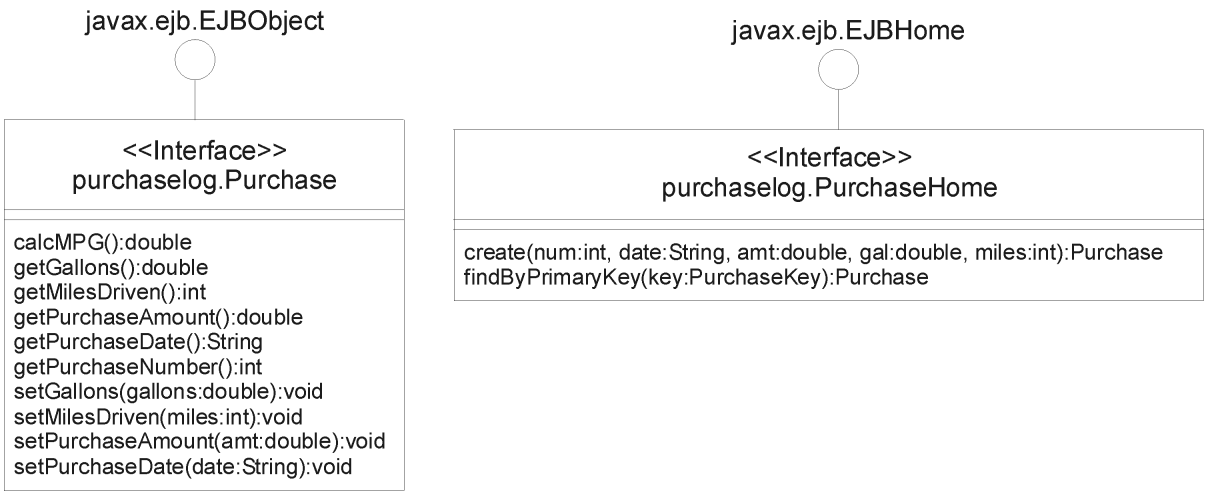

Figure 1: Part 1 UML Class Diagram

The *Purchase* interface represents a fuel purchase and defines get/set methods for the purchase information. It also defines the *calcMPG* logic method that returns the miles per gallon for the purchase. The *getPurchaseNumber* method returns the unique purchase number integer for a given purchase -- this will act as the primary key. Though the figure doesn't show it, we will provide you with a class to represent the primary key.

The *PurchaseHome* interface represents the home for this container-managed entity bean. It defines a simple "create" method as well as the standard "finder" method the locates a Purchase given its primary key.

Objectives:

- To write a CMP entity Enterprise JavaBean (Part 1)
- To write a session bean that uses an entity bean (Part 2)

Part 1: Writing a CMP Entity Bean

In this part, you write a simple CMP entity bean that represents a gas purchase and its accessory types (home, key and implementation).

Steps:

_1. Ask your instructor for the following information:

BEA Installation directory: _____

Lab Installation directory: _____

If your machine has the standard lab setup, the directories will be something like:

```
BEA Installation directory:    c:\java\bea
Lab Installation directory:    c:\j2eeclass
```

_2. First you need to create the database that your beans will use. Follow these steps:

a. Use Windows Explorer to navigate to the **{BEA Installation directory}/weblogic81/common/eval/pointbase/tools** directory. Start the database server by clicking on **startPointBase.cmd**. Leave the PointBase server window open.

b. PointBase includes a simple GUI to let you perform administrative tasks. To start the GUI, click on **startPointBaseConsole.cmd** in Windows Explorer.

c. In the *Connect to Database* dialog, enter the following:

```
Driver:              com.pointbase.jdbc.jdbcUniversalDriver
URL:                 jdbc:pointbase:server://localhost/ejbpurchase
User:                javauser
Password:            javadude
```

Then select the **Create New Database** button and then press OK.

_3. Now you need to configure WebLogic container so it knows about the new database. Follow these steps:

a. Start the container and wait for it to be ready.

b. Open a Java enabled web browser and enter this URL:

```
http://localhost:7001/console
```

This is the *Administrator Console* for WebLogic. Login with the username and password provided by the instructor (typically "administrator" for both).

Wait for the embedded Java applet to display a tree in the left window pane.

c. In the left window pane, expand *mydomain* - *Services* - *JDBC*, then click on *Connection Pools*. If any pools have already been defined, you will see a table listing them. If there are none, you can proceed to the next step where you will define a new connection pool, but if there are any listed, you will need to un-deploy them.

To un-deploy a pool, in the table, notice the *Deployed* column. For the first pool that with a *true* value, click on the pool's entry in the *name* column, then click the *Target and Deploy* tab. Uncheck the box for *myserver*, then press the Apply button.

Repeat the above for all currently deployed pools. You can redisplay the Connection Pool table by clicking on *Connection Pools* in the tree display in the left window pane.

d. To define a new pool, click on *Configure a New JDBC Connection Pool*. Select a *Database Type* of **PointBase**, and driver of **PointBase's Driver (Type 4) Versions:4X**.

Warning: Do not choose the *4XA* driver type.

Press Continue.

e. On the *Configure a JDBC Connection Pool* page, enter the following:

```
Name:                PurchasePool
Database Name:       ejbpurchase
Hostname:            localhost
Port:                9092
Database User Name: javauser
Database Password:  javadude
```

Press *Continue*.

f. On the *Test database connection* page, press the **Test Driver Configuration** button -- you should see a "Connection successful" message.

Press the **Create and deploy** button.

g. In the left window pane tree, click on the *Data Sources* entry and then click on *Configure a new JDBC Data Source*.

h. On the *Configure a JDBC Data Source* page, enter the following:

```
Name:                PurchaseDataSource
JNDI Name:           PurchaseDataSource
```

Press *Continue*.

i. On the *Connect to connection pool* page, select **PurchasePool**, then press Continue.

j. On the *Target the data source* page, ensure that the *myserver* entry is checked (selected), then press Create.

k. You can then close the Administrator's Console.

_4. Create a file named **Purchase.java** and save it in the **{Lab Installation Directory}/ejblab05/ejb** directory. In this file, write the *Purchase* interface as described in the UML diagram above. Put the interface in the *purchaselog* package.

_5. Examine the completed **PurchaseKey.java** file that implements the primary key. Make sure you understand how it works.

_6. Create a file named **PurchaseHome.java**, and in it, write the *PurchaseHome* interface. Be sure to put the interface in the *purchaselog* package.

_7. To save you typing, we have provided you with a partially completed bean implementation. Edit **PurchaseBean.java**, and in it, complete the *PurchaseBean* class. Define public *abstract* get/set methods for each of the properties specified in the remote interface:

- *purchaseNumber* A unique integer that identifies a particular purchase
- *purchaseDate* A String that contains the date when the fuel was purchased
- *gallons* The number of gallons purchased (double)
- *milesDriven* The number of miles driven (int)
- *purchaseAmount* The dollar amount of the purchase (double)

You will need to add the following methods:

- *ejbCreate* Corresponds to the *create* method in the home interface. In this method, you should call the abstract *set* methods to assign the bean's property values from the arguments passed to this method.
- All "business logic" methods from the remote interface, i.e. *calcMPG*. This method should throw the *EJBException* if the *gallons* property is zero, else return the calculated miles per gallon.

_8. Edit the partially completed **META-INF/ejb-jar.xml** deployment descriptor and complete it. To save you typing, we have provided you with most of the elements -- you just need to fill in the "xxxxx" markers.

Note: Please make sure that the *ejb-name* is **Purchase**.

_9. Edit the fully completed **META-INF/weblogic-ejb-jar.xml** file and examine this WebLogic-specific deployment descriptor to see if you can make sense of it.

_10. Edit the fully completed **META-INF/weblogic-cmp-rdbms-jar.xml** file and examine this WebLogic-specific deployment descriptor to see if you can make sense of it.

Note how it specifies the data-source name, the table name and maps CMP fields to columns in the table. Also notice that it specifies that the container should create the table automatically.

_11. Open a command prompt and change to the **{Lab Installation Directory}/ejblab05** directory. Note that this is the parent of the **ejb** directory.

_12. Look in the **{Lab Installation Directory}/cmp** directory for a file named **build.xml** -- it contains the Ant commands to build and deploy -- take a look at it in your editor and see if you can make sense of it. Note how it builds a JAR named **ejblab05.jar** to hold your server-side components.

To build and deploy your server-side code, in your command prompt window, type:

```
ant deploy -emacs
```

Be sure that you had a successful build and deploy and fix any compile errors before continuing. Be sure to monitor the container's window for deployment errors.

Part 2: Standalone Java Client

Steps:

_1. Create a file named **PurchaseClient.java** in the **{Lab Installation Directory}/ejblab05/client** directory and in it, write a standalone Java client program with a *main* method that:

- Creates an initial context
- Retrieves the home reference
- Casts the reference to the correct type
- Creates a *Purchase*, using the *create* method from the home interface
- Displays the miles per gallon to the console

To save typing, you can copy the client (or solution) from the last lab. Be sure to modify the client so that it uses the correct names for your server types. **Note:** The client program class does not need to be in a package.

_2. In your command window, make sure that the current directory is the **{Lab Installation Directory}/ejblab05** directory (i.e. the parent of the **client** directory). Then use Ant to build the client:

```
ant build-client -emacs
```

Be sure to correct any errors before continuing.

_3. Use Windows Explorer to copy the **jndi.properties** file from the previous lab.

_4. In your command window, make sure that the current directory is the **{Lab Installation Directory}/ejblab05** directory (i.e. the parent of the **client** directory). Then use Ant to run the client:

```
ant run-client -emacs
```

Do you see the miles-per-gallon message? If you wrote any System.out.println**s** in the EJB implementation, you should see them in the container's console.

_5. Try running the client again. Did the client get a duplicate-key exception? It should! Why?

_6. To examine the database, start the PointBase GUI like you did in an earlier step (open the database instead of creating a new one).

In the GUI's leftmost pane, expand the tree under *SCHEMAS* until you see the *purchase* table. You can expand the *purchase* table's *COLUMNS* entry to see the table schema. Does the table's schema match your EJB's CMP fields?

Then, in the upper right window pane, enter:

```
SELECT * FROM purchase
```

Then press the Execute button -- you should see the data for the purchase.

Part 3: Using Session Beans and Entity Beans Together

In this part, you will add to your program by writing a stateless session bean that will make it easier to write clients of the Purchase entity bean. This will help demonstrate how to combine session and entity beans in an

application. Here is a UML diagram that shows the session bean interface:

Figure 2: Part 2 UML Class Diagram

Note that clients can call the *addPurchase* method without having to provide a purchase number -- the *PurchaseLog* session bean will calculate the purchase number.

Steps:

_1. Create a file named **PurchaseLog.java** in the **{Lab Installation Directory}/ejblab05/ejb** directory, and in it, write the *PurchaseLog* interface. Be sure to put the interface in the *purchaselog* package.

_2. Create a file named **PurchaseLogHome.java**, and in it, write the *PurchaseLogHome* interface. Be sure to put the interface in the *purchaselog* package. Remember that this is a stateless session bean!

_3. Create a file named **PurchaseLogBean.java**, and in it, write the *PurchaseLogBean* class that implements the PurchaseLog interface. Be sure to put the class in the *purchaselog* package. This class should implement the required EJB session-bean lifecycle methods (with empty method bodies except for *setSessionContext*) and the "business logic" methods from the PurchaseLog interface. The next few steps give some hints on writing the bean's methods.

Note: Don't forget *ejbCreate*! Also remember that you can copy code from earlier labs to save typing.

_4. In the *setSessionContext* method, write code to retrieve an *InitialContext* and store its reference in a private field of type *Context* named *jndiContext*. Use code like:

```
try
{
    jndiContext = new InitialContext();
}
catch ( Exception exc )
{
    throw new EJBException ( exc );
}
```

Note: To save typing, you can copy and paste this text from the **snippets.txt** file in the **{Lab Installation Directory}/ejblab05/ejb** directory.

_5. Write a private method that retrieves the home for the *Purchase* entity bean. This method will be useful since you can call it multiple times. Here's the code for the method (you can copy it from **snippets.txt)**:

```
private PurchaseHome getHome()
{
    try
    {
        Object o = jndiContext.lookup (
            "java:comp/env/ejb/Purchase" );

        return (PurchaseHome)
            PortableRemoteObject.narrow (
                o, PurchaseHome.class );
    }
    catch ( Exception exc )
    {
        throw new EJBException ( exc );
    }
}
```

_6. Write a private method that returns a random number for the purchase number. Here's the code for the method (you can copy it from **snippets.txt)**

```
private int getRandPosInt()
{
    Random rand = new Random();
    int i = 0;
    while ( i == 0 )
    {
        i = rand.nextInt();
        if ( i < 0 )
            i = -i;
    }

    return i;
}
```

Note: The *Random* class in the *java.util* package, so you should import that package. You should also import the *javax.naming* and *javax.rmi* packages.

_7. In the *addPurchase* method, write code that:

- Retrieves a random number for the purchase number by calling the private method you just wrote
- Retrieves a Purchase bean home reference by calling the private method you wrote earlier
- Calls the home's *create* method, passing the calculated purchase number and other method arguments
- Returns the newly created Purchase reference

If the creation fails, throw the *EJBException.*

_8. In the *deletePurchase* method, write code that:

- Retrieves a Purchase bean home reference
- Retrieves a reference to the specified Purchase by calling *findByPrimaryKey* on the home
- Calls the *remove* method on the Purchase reference

If the deletion fails, throw the *EJBException*.

_9. Edit the **ejb-jar.xml** deployment descriptor, and add a <session> element for the *PurchaseLog* session bean (insert it immediately before the <entity> element). To save typing, you can copy this element from one of the earlier stateless session bean labs.

Note: Please use an *ejb-name* of **PurchaseLog**.

Then, immediately before the closing </session> tag, insert:

```
<ejb-ref>
   <ejb-ref-name>ejb/Purchase</ejb-ref-name>
   <ejb-ref-type>Entity</ejb-ref-type>
   <home>purchaselog.PurchaseHome</home>
   <remote>purchaselog.Purchase</remote>
   <ejb-link>Purchase</ejb-link>
 </ejb-ref>
```

This element ensures that the session bean will be able to look up the entity bean's home reference in JNDI.

_10. Edit the **weblogic-ejb-jar.xml** deployment descriptor and add the following immediately before the existing *weblogic-enterprise-bean* element:

```
  <weblogic-enterprise-bean>
    <ejb-name>PurchaseLog</ejb-name>
    <jndi-name>PurchaseLog</jndi-name>
  </weblogic-enterprise-bean>
```

_11. Return to the command window in which you ran Ant in the last part. Make sure the current directory is the **{Lab Installation Directory}/ejblab05** directory. Then use Ant to compile, rebuild and deploy as before:

```
ant deploy -emacs
```

Be sure that you had a successful build and deploy and fix any compile errors before continuing. Be sure to monitor the container window for deployment errors.

Part 4: Client for Session Bean

Steps:

_1. Create a new file named **PurchaseLogClient.java** in the **client** directory, and in it, write code that:

- Retrieves a reference to the *PurchaseLog* home

- Retrieves a reference to a *PurchaseLog* stateless session bean by calling the home's *create* method
- Calls the PurchaseLog's *addPurchase* method to create a new Purchase and retrieve its reference
- Display the new Purchase's purchase number to the console

To save typing, you can copy code from the **PurchaseClient.java** you wrote in the last part.

_2. Return to the command window in which you ran Ant. Make sure the current directory is the **{Lab Installation Directory}/ejblab05** directory (the parent of the **client** directory). Use Ant to compile the new client:

```
ant build-client -emacs
```

_3. In your command window, make sure that the current directory is the **{Lab Installation Directory}/ejblab05** directory (i.e. the parent of the **client** directory). Then use Ant to run the new client:

```
ant run-client-part2 -emacs
```

Do you see the purchase number? What happens if you run the client more than once? Does that make sense?

_4. Stop the container and start the PointBase GUI as before. Then like before, use the SELECT command to display all of the rows from the purchase table. Do you see the new rows?

Optional:

Optional labs are for students that wish to explore topics further with minimal guidance from the lab write-up (and no solution!).

_1. Add a new method to the *Purchase* home interface with the following signature:

```
Collection findAll() throws RemoteException, FinderException;
```

This is a finder method. Recall that for CMP, finder methods are implemented by the container. So the good news is that you won't have to modify the PurchaseBean implementation! However, you should modify the PurchaseLog interface and PurchaseLogBean implementation to use the new finder method. In other words, modify the PurchaseLog interface so that the client can retrieve a *Collection* of all Purchases. The PurchaseLogBean implementation can simply call the finder method on the Purchase home. Then modify the client to use the new method to display all Purchases to the console. **Hint:** The Collection interface is in the *java.util* package.

You will need to add the following lines to the **ejb-jar.xml** file immediately after the last *cmp-field* definition:

```
 <query>
   <query-method>
      <method-name>findAll</method-name>
      <method-params/>
   </query-method>
   <ejb-ql><![CDATA[SELECT OBJECT(a) FROM PurchaseBean AS a]]></ejb-ql>
 </query>
</entity>
```