Exercises for Chapter 6

Exercise 6.1:

This exercise demonstrates a simple Customer EJB. You will be creating a new application archive for each of the examples in this chapter. To facilitate easy manipulation of the EJBs you will be provided with pre-packaged *.jars* for the entity beans. Chapter 6 progressively demonstrates management of declarative security, dependant objects and simple one-to-one relationships between entity beans with CMP 2.0.

Building the application for Exercise 6.1

1. Download the *ex06-1.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\ex06_1*, referred to as *$EX06_1_HOME*.
4. Open a command prompt and type `Ant` or `Ant -buildfile build.xml`

This should create *$EXAMPLES_HOME\build\ex06_1* directory and compile the required files.

$EXAMPLES_HOME\pre-built\ex06_1 directory contains pre-packaged client application jars for this exercise

Database schema for the Cabin EJB

This exercise will create the `CustomerBeanTable` as part of the deployment process.

Building the Application Archive for Ex06_1

1. Run the deployment tool by typing `c:\>deploytool` at the command prompt.
2. Create a new application and call it *Titan06_1App.ear*.

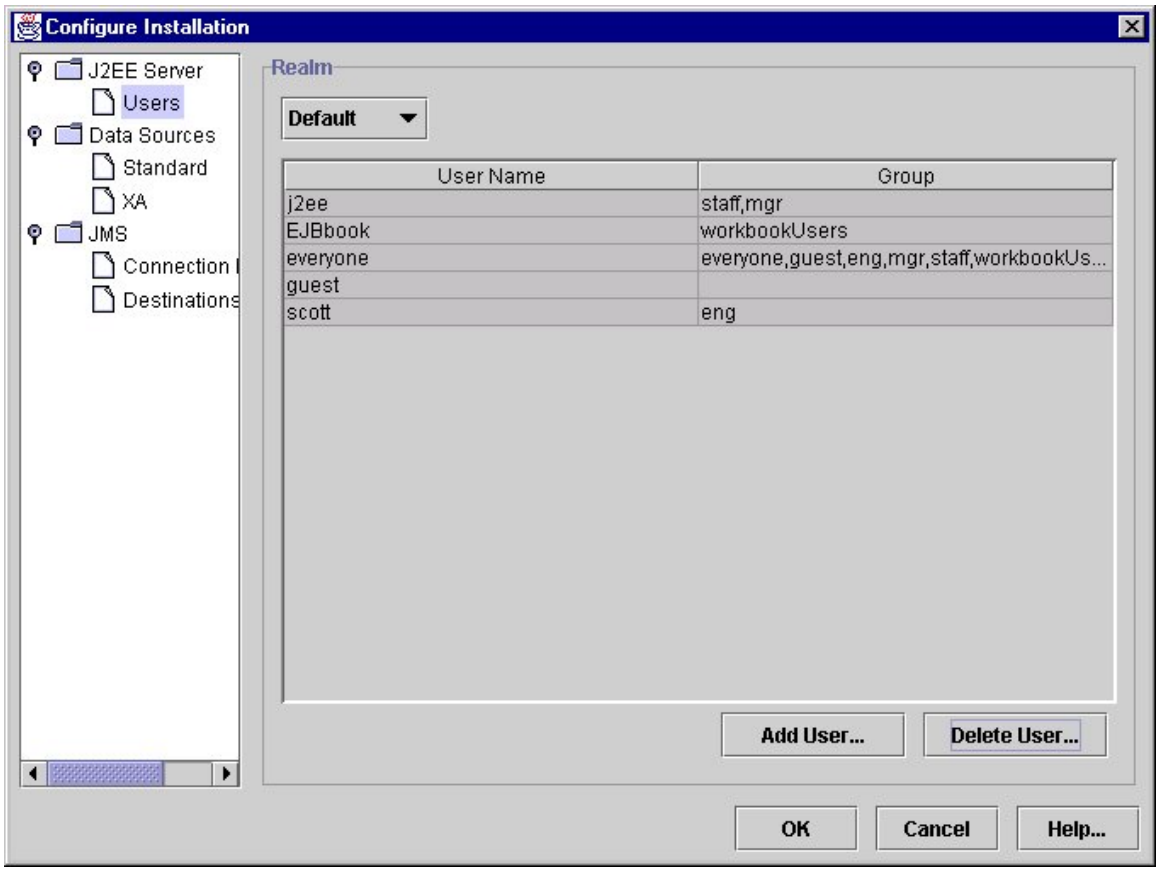
A pre-built application is available in *$EXAMPLES_HOME/ears* directory.

3. Create a new EJB and name it *CustomerEJB*. As explained in Chapter 4 create the simple CustomerEJB by mapping the container-managed fields and generating the default SQL.
4. Select **File→Add to Application→ApplicationClient Jar** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex06_1/client_61.jar* and add it to the application. You can also create the application client using **File→New→ApplicationClient** option.

This example demonstrates the setup of the declarative security model used in J2EE.

1. First you need to create a user with a specified role in the J2EE application server environment. Select **Tools→Server Configuration** to open up the server configuration window.

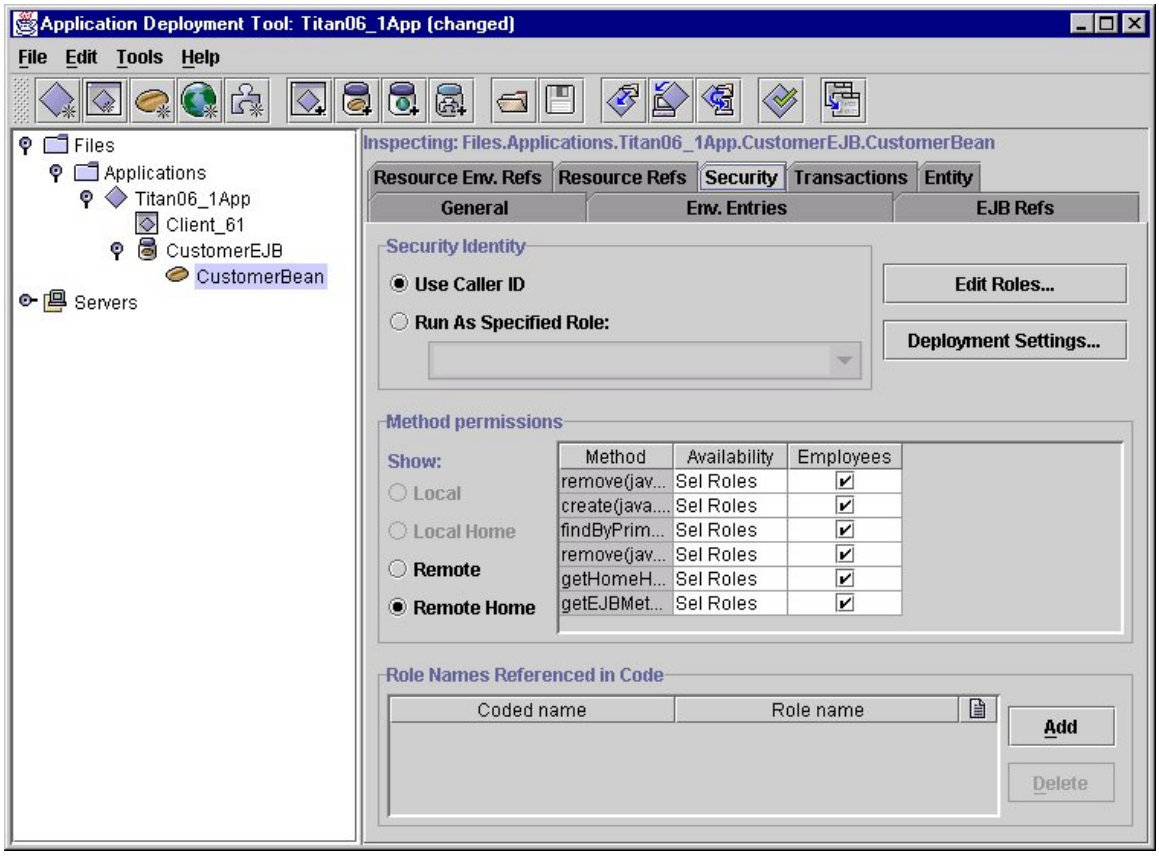
Figure 1: Server Configuration



2. Add a user **Client_61** with password **client_61** by clicking on the **Add User** button.
3. Click on the **Edit Groups** button to create a group called **J2EEUsers**.
4. Associate the **Client_61** user with **J2EEUsers** group from the available groups. This will create a user associated with the Employees group. You need to shut down and re-start the J2EE server in order for the configuration changes to take effect.

Now it is time to associate the Customer EJB with a user in a role of Employees. Users or Groups that are known as Principals should be associated with a role to execute operations on the EJB.

Figure 2 Role association for the EJB



1. Select the *Customer* EJB in the deployment tool and select the **Roles** tab. Add a new role **Employees** to access the EJB.
2. Select the CustomerBean EJB in the *Customer* EJB jar and click on the **Security** tab.
3. Associate the remote and remote home methods with the **Employees** role.
4. Select the Application node for Titan06_1App and click on the **Security** tab.
5. Add the **J2EEUsers** group to be associated with the **Employees** role.

This completes the configuration of the Titan06_1App. Users belonging to the J2EEUsers group are associated with the Employees role and can access the Customer EJB.

6. 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.

7. 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 and Run the Ex06_1 client applications.

This exercise consists of one client application. This example takes command line arguments in multiples of three. This example creates Customer EJBs based on the command line arguments. The output shown below demonstrates the use of authentication for the privileged user **Client_61** belonging to the group **J2EEUsers** and having a role of **Employees**. We provide a text-based authentication for clear text password.

```
C:\EJBBOOK\build>runclient -client Titan06_1App.ear -name Client_61
-textauth 777 Prasad Muppirala 888 Greg Nyberg
Initiating login ...
Username = null
Enter Username:Client_61
Enter Password:client_61
Binding name:`java:comp/env/ejb/CustomerHomeRemote`
777 = Prasad Muppirala
888 = Greg Nyberg
Unbinding name:`java:comp/env/ejb/CustomerHomeRemote`
```

This example creates two rows in the `CustomerBeanTable` and removes them at the end of the program run. You can edit the client program and test different steps during the process.

Examine and run the Web Client for Client_61 program

This example does not provide a web-based client. It is left as an exercise for the user to create a simple HTML form that can accept a primary key value, first name and last name and be submitted to create the Customer EJB. Security settings for the web-based applications also need to be set in the application.

Exercise 6.2:

This exercise demonstrates the use of the dependent value object `Name` that is used to encapsulate the first and last name of the *Customer* EJB. Bulk accessors can also be used to perform the set operations. The flexibility gained by providing a data structure for the value objects is such that they can be serialized across the wire and be accessible for the data values of the EJB.

Building the application for Exercise 6.2

1. Download the *ex06-2.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 to *$EXAMPLES_HOME* using *WinZip* or *.jar* with xvf flags. Example *c:\>EJBBook* directory.
3. Change directory to *$EXAMPLES_HOME\src\ex06_2*, referred to as *$EX06_2_HOME*.
4. Open a command prompt and type `Ant` or `Ant -buildfile build.xml`

This should create *$EXAMPLES_HOME\build\ex06_2* directory and compile the required files. *$EXAMPLES_HOME\pre-built\ex06_2* directory contains pre-packaged client application *jar*s for this exercise.

5. As shown in the exercise 6.1 create the *Titan06_2App* application components: **CustomerEJB**, **Client_62** and **Titan06_2Web** client with the context URL of *titan62*.

Database schema for the Cabin EJB

This exercise will use the `CustomerBeanTable` created in Exercise 6.1

Building the Application Archive for Ex06_2

1. Run the deployment tool by typing `c:\>deploytool` at the command prompt.
2. Create a new application and call it *Titan06_2App.ear*.

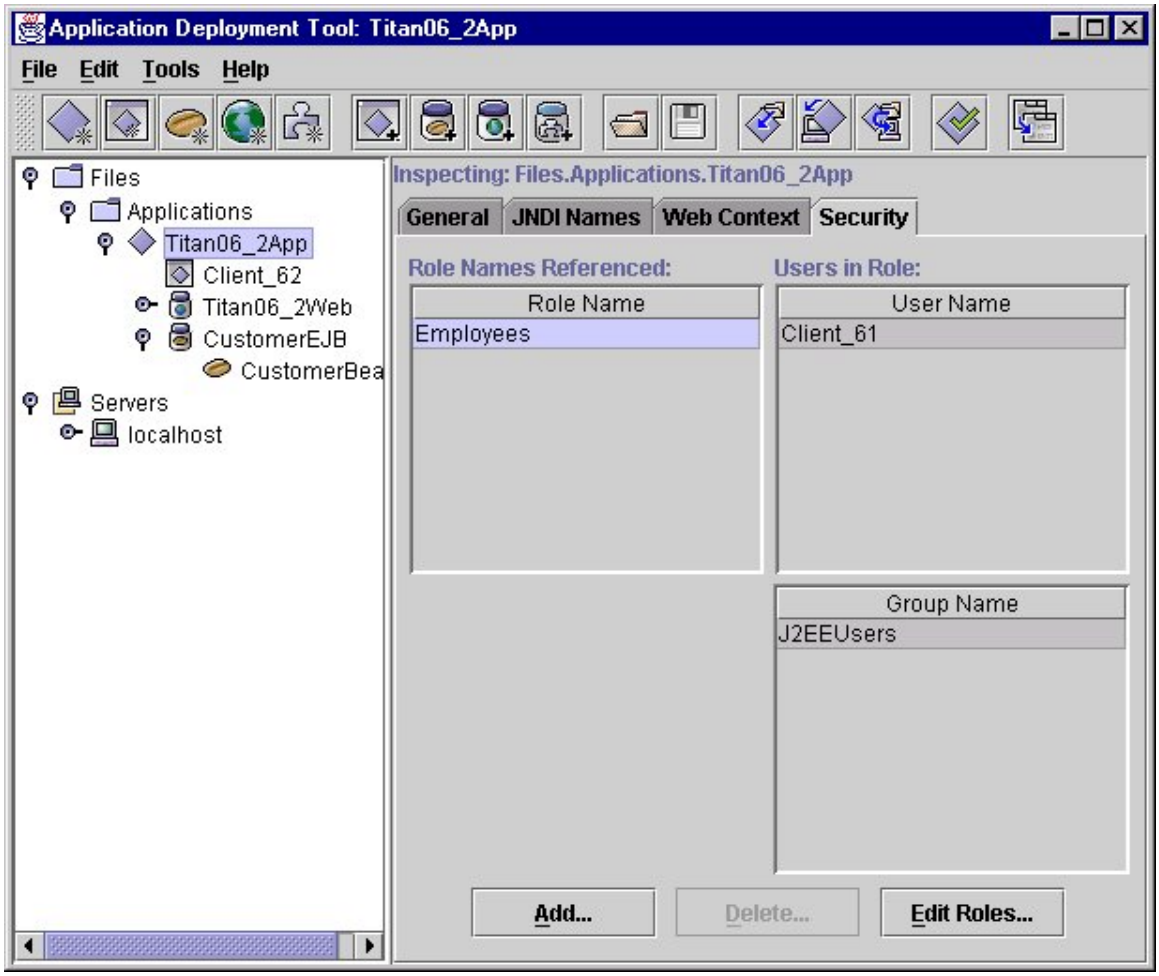
A pre-built application is available in *$EXAMPLES_HOME/ears* directory.

3. Create a new EJB and name it *CustomerEJB*. As explained in Chapter 4 create the simple CustomerEJB by mapping the container managed fields and generating the default SQL.
4. Select **File→Add to Application→ApplicationClient Jar** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex06_2/client_62.jar* and add it to the application. You can also create the application client using the **File→New→ApplicationClient** option.

5. Select **File→Add to Application→Web war** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex06_2/client_62.jar* and add it to the application. You can also create the application client using **File→New→Web war** option.

6. In this exercise the `get` and `set` methods for the `lastName` and `firstName` `cmp` fields are not made available from the remote object. These methods need to be defined as public for the container to map these accessors to the fields derived for persistence storage.

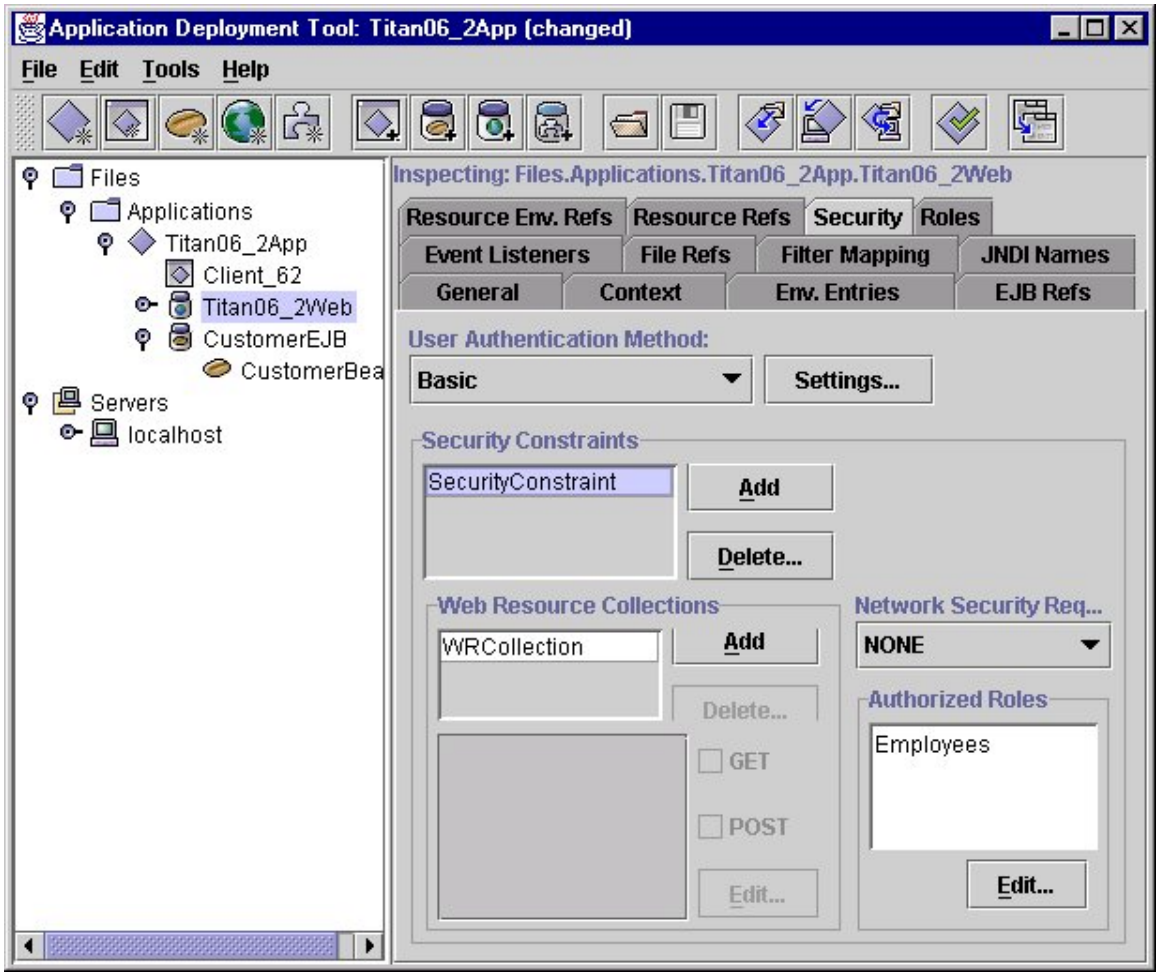
Figure 3: Role assignment to User/Group



7. Make sure the application security rules are set.

8. Validate the existence of the Client_61 user within the J2EEUsers group by viewing the Server configuration.
9. Select the Titan06_2App node and add the J2EEUsers group to the Employees role.
10. Verify the security constraints for the web application. Set the security to basic authentication.

Figure 4: Security constaints configuration for Web application



11. Assign access to *Client_62.jsp* only to Principals assigned with the role of Employees. As mentioned earlier, J2EE defines Users and Groups as Principals. Groups can contain other Groups and Users. J2EE components define access privileges with respect to roles. Each Principal can be assigned roles based on which the container will allow access to a

Principal for a given component. This assignment of roles to one or more Principals is the responsibility of the security interface provided by each server. The role assignment for accessing a component or a method in a component is typically defined declaratively through the packaging and deployment mechanism of each server.

12. Create a `Security Constraint` and add a Web Resource collection for the web application. Assign the role access for the Web Resource Collection. A Web Resource Collection is a set of HTML and JSP pages that will have permissions set for access to the Principals assigned for the specific role.
13. Verify the *CustomerBean* EJB has access defined for selected roles choice in the drop-down list and Employees role is checked for all the remote and remoteHome methods.
14. Run the application through the verifier for any errors.
15. Deploy the application and select the **return client jar** check-box.
16. Assign Titan62 and the root context for the *War* file.

Examine and run the Client application

The client application for this exercise makes use of the `Name` dependent value class to create the *Customer* EJB. The output shown is for the client application.

```
C:\EJBBOOK\build>runclient -client Titan06_2App.ear -name Client_62
-textauth
Initiating login ...
Username = null
Enter Username:Client_61
Enter Password:client61
Binding name:`java:comp/env/ejb/CustomerHomeRemote`
Creating customer for use in example...
Getting name of customer using getName()..
1 = Richard Monson
Setting name of customer using setName()..
Getting name of customer using getName()..
1 = Richard Monson-Haefel
Removing customer...
Unbinding name:`java:comp/env/ejb/CustomerHomeRemote`
```

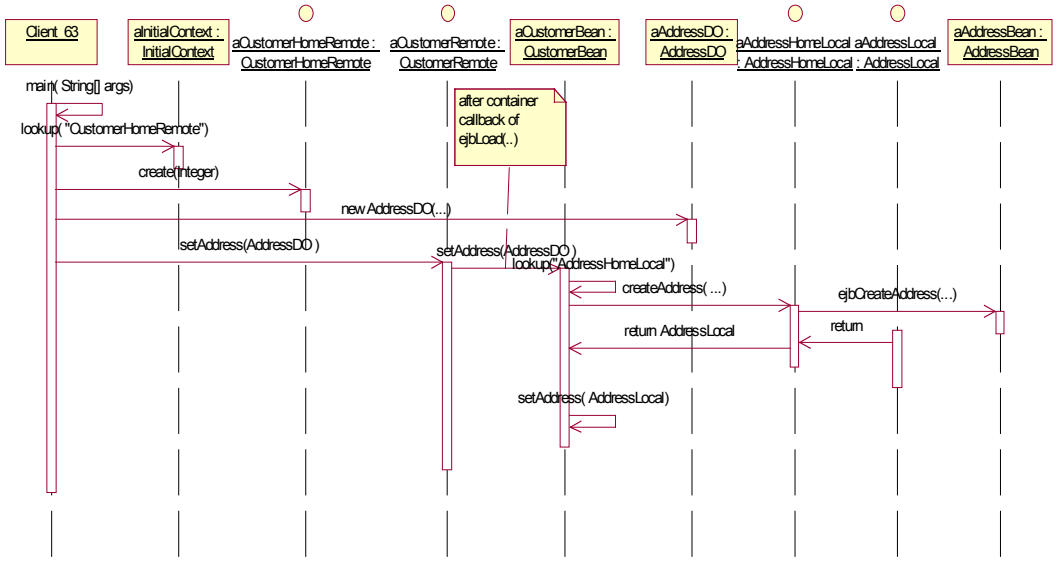
This example demonstrates the dependent value object usage pattern.

Exercise 6.3:

This exercise extends the dependant value object concept along with the usage of local interfaces that are new to CMP 2.0. This exercise introduces another Entity Bean Address EJB, that is accessed via a local interface from *Customer* EJB. This exercise also makes use of relationship fields (cmp) as defined by CMP 2.0.

The following interaction diagram shows the remote client accessing the `CustomerEJB` and in turn the `CustomerEJB` creates an `AddressEJB`.

Figure 5: Interaction diagram showing client access to EJB



This interaction shows only partial execution threads and in most cases doesn't show the container call-backs. This is just an indication of the Client creating a *CustomerEJB* and `AddressDO,` and setting the `AddressDO` on customer which creates the *AddressEJB*.

Explanation of Example code

This example contains two EJBs, *Customer* EJB and *Address* EJB. It also uses a serializable value object `AddressDO`. Client access is limited to *CustomerEJB* through remote interfaces. *Address* EJB is available via the local interfaces. We have established a naming convention for distinctly

identifying the remote and local interfaces. The following table shows the corresponding files of both the EJBs.

Definition	Customer EJB	Address EJB
Bean Class	*CustomerBean.java*	*AddressBean.java*
Home Interface (`javax.ejb.EJBHome`)	*CustomerHomeRemote.java*	N/A
Home Interface (`javax.ejb.EJBLocalHome`)	N/A	*AddressHomeLocal.java*
Remote Interface (`javax.ejb.EJBObject`)	*CustomerRemote.java*	N/A
Local Interface (`javax.ejb.EJBLocalObject`)	N/A	*AddressLocal.java*

Let us get into the interesting methods of the source files.

AddressBean.java

Address Bean is a simple entity bean with one `create` method.

```
public Integer ejbCreateAddress
             (Integer addressID,String street, String city,
              String state,  String zip )
   throws javax.ejb.CreateException
    {
         setId(addressID);
         setStreet(street);
         setCity(city);
         setState(state);
         setZip(zip);
         return null;
    }
```

Note: This method takes in primary key value as an argument of **addressID**. J2EE™ SDK does not provide automatic key generation capability. We could have taken an approach of creating a sequence table and as part of this method accessed the sequence value either directly via JDBC or obtained the sequence value via another stateless session bean. Some EJB containers like Weblogic provide sequence generation capability as part of their extension. We take a simple approach of passing the **addressID** as one of the arguments.

AddressLocal.java

This interface extends `javax.ejb.EJBLocalObject` and provides access to all the accessors for the persistence fields and other methods available for the caller via the local interface.

```
public interface AddressLocal extends javax.ejb.EJBLocalObject {
```

```
  public String getStreet();
  public void setStreet(String street);
  public String getCity();
  public void setCity(String city);
  public String getState();
  public void setState(String state);
  public String getZip();
  public void setZip(String zip);
  public void setId(Integer id);
  public Integer getId();
}
```

These methods do not throw `RemoteException` as they are accessed locally rather than via a remote protocol like JRMP or RMI/IIOP.

AddressHomeLocal.java

This is the Home Interface that returns a local reference to the EJB and contains the typical matching `create` and `finder` methods.

// *Address* EJB's local home interface

```
public interface AddressHomeLocal extends javax.ejb.EJBLocalHome {

  public AddressLocal createAddress(
                    Integer addressID,String street, String city,
                    String state,  String zip )
                    throws javax.ejb.CreateException;

  public AddressLocal findByPrimaryKey(Integer primaryKey)
  throws javax.ejb.FinderException;
}
```

AddressDO.java

This class implements `java.io.Serializable` interface. This is similar to the `Name` class used in previous exercises and behaves as a dependent value object. This can be passed to the clients accessing the EJB via a distributed protocol and is passed-by-value.

This class encapsulates the intrinsic attributes by making them `private` and provides `public` accessors.

CustomerRemote.java

This interface extends the `javax.ejb.EJBObject` and provides methods that are publicly accessible by any client. Also this interface provides access to the serialized data access object `AddressDO` rather than a reference to *Address* EJB.

```
public interface CustomerRemote extends javax.ejb.EJBObject {
```

```
    public void setAddress(Integer addressID, String street, String
                           city, String state, String zip)
          throws RemoteException, CreateException, NamingException;

    public void setAddress(AddressDO address)
          throws RemoteException, CreateException, NamingException;

    public AddressDO getAddress() throws RemoteException;

     public Name getName() throws RemoteException;
     public void setName(Name name) throws RemoteException;

    public boolean getHasGoodCredit() throws RemoteException;
    public void setHasGoodCredit(boolean flag) throws
                                          RemoteException;

}
```

The implementation of `get` and `set` for the container does not generate an Address. We will examine the code that a developer has to write in order to implement these methods.

CustomerBean.java

This class implements the `javax.ejb.EntityBean` interface and provides abstract `get` and `set` methods for its container managed fields. The `getAddress()` and `setAddress(…)` methods return `AddressDO` and `setAddress(AddressDO)`.

```
    public AddressDO getAddress(){

         AddressLocal addrLocal = this.getHomeAddress()
         Integer addressID = addrLocal.getId();
         String street = addrLocal.getStreet();
         String city = addrLocal.getCity();
         String state = addrLocal.getState();
         String zip = addrLocal.getZip();
         AddressDO addrValue = new
                     AddressDO(addressID,street,city,state,zip);
         return addrValue;
     }

     public void setAddress(AddressDO addrValue)
          throws CreateException, NamingException {

         String street = addrValue.getStreet();
         String city = addrValue.getCity();
         String state = addrValue.getState();
         String zip = addrValue.getZip();
```

```
            Integer addressID = addrValue.getId();

             setAddress(addressID, street,city,state,zip);
        }
```

Even though `CreateException` is not needed, the J2EE™ SDK complains if the methods do not throw the `CreateException`.

Building the application for Exercise 6.2

1. Download the *ex06-3.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 to *$EXAMPLES_HOME* using *WinZip* or *jar* with xvf flags. Example *c:\> EJBBook* directory.
3. Change directory to *$EXAMPLES_HOME\src\ex06_3*, referred to as *$EX06_2_HOME*.
4. Open a command prompt and type `Ant` or `Ant -buildfile build.xml.`

This should create *$EXAMPLES_HOME\build\ex06_2*3directory and compile the required files.

$EXAMPLES_HOME\pre-built\ex06_3 directory contains pre-packaged client application *jars* for this exercise.

5. As shown in the previous exercise create the *Titan06_3App* application components. *CustomerEJB, AddressEJB, Client_63* and *Titan06_3Web* client with the context URL of *titan63*.

Database schema for the Customer and Address EJB

The deployment process will create the required tables. Make sure that the `CustomerBeanTable` is removed.

Building the Application Archive for Ex06_3

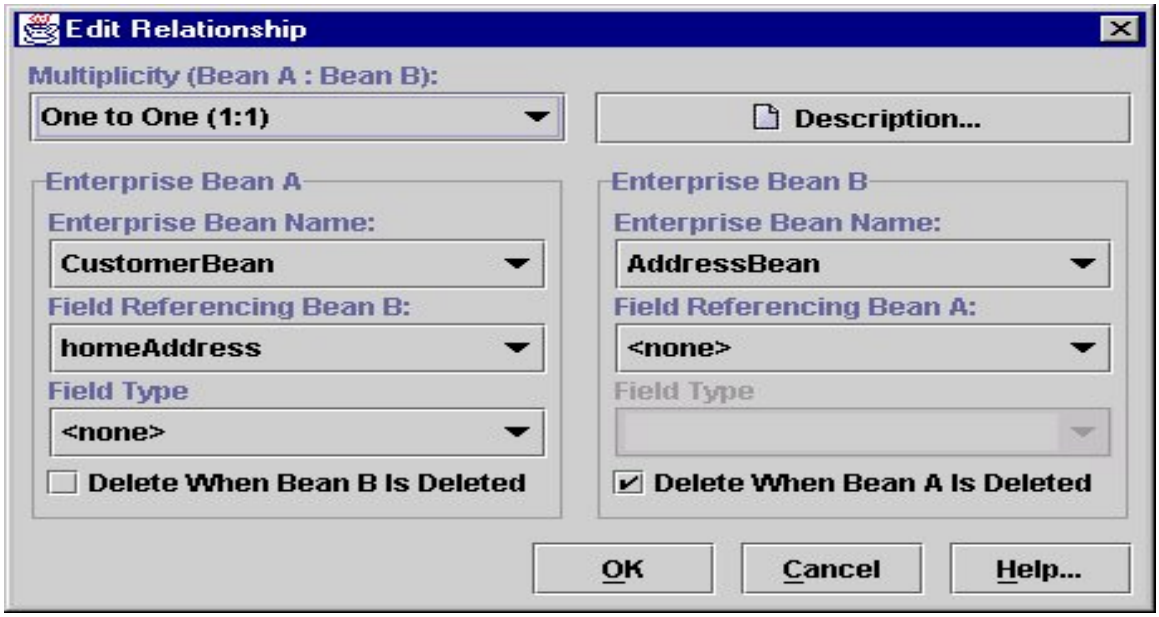
1. Run the deployment tool by typing `c:\>deploytool` at the command prompt.
2. Create a new application and call it *Titan06_3App.ear*.

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

3. Create a new EJB and name it *CustomerEJB*. As explained in Chapter 4 create the simple *CustomerEJB* by mapping the container managed fields and generating the default SQL. Do not map the `homeAddress` field.
4. Create a new EJB in the same *.jar* and name it *AddressEJB*. Similar to *Customer* EJB select all the persistent fields to generate the SQL.

5. It is time to setup the EJB references in *Customer* EJB so that *Customer* EJB can access *Address* EJB. Select the **EJB Refs** tab for the *Customer* EJB and add the following
   - Coded Name: **ejb/AddressHomeLocal**
   - Type: **Entity**
   - Interfaces: **Local**
   - Home Interface: **com.titan.customer.AddressHomeLocal**
   - Local/Remote Interface: **com.titan.customer.AddressLocal**
6. You also need to set up the `cmr` field for a One-One relationship. This can be done by selecting the EJB *jar* node and selecting the **Relationships** tab. Click the **Add** button and set up the relationship.

Figure 6: One-One relationship between Customer and Address



This will set up the `cmr` field for the *Customer* EJB as the *Address* EJB.

7. Select **File→Add to Application→ApplicationClient Jar** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex06_3/client_63.jar* and add it to the application. You can also create the application client using the **File→New→ApplicationClient** option.

8. Select **File→Add to Application→Web war** from the menu. Select from the *$EXAMPLES_HOME/pre-built/ex06_3/client_63.jar* and add it to the application. You can also create the application client using the **File→New→Web war** option.
9. Set up the security credentials similar to Exercise 6.2. Add the role Employees to the EJBs and select all the methods for the selected role. Assign the principal J2EEUsers to this role for the Client and Web Application.
10. Run the application through the verifier to make sure there are no failed tests.
11. Delete the `CustomerBeanTable` from the database.
12. Deploy the application *Titan06_3App.ear* to the server. This completes the deployment process.

Examine and run the Client application

The client application for this exercise makes use of the `AddressDO` dependent value class to create the Address EJB when the Customer EJB executes the `setAddress(…)` method. The output shown is for the client application.

```
C:\EJBBOOK\build>runclient -client Titan06_3App.ear -name Client_63
-textauth
Initiating login ...
Username = null
Enter Username:Client_61
Enter Password:client61
Binding name:`java:comp/env/ejb/CustomerHomeRemote`
Creating Customer 1..
Creating AddressDO data object..
Setting Address in Customer 1...
Acquiring Address data object from Customer 1...
Customer 1 Address data:
1010 Colorado
Austin,TX 78701
Creating new AddressDO data object..
Setting new Address in Customer 1...
Customer 1 Address data:
1600 Pennsylvania Avenue NW
DC,WA 20500
Removing Customer 1...
Unbinding name:`java:comp/env/ejb/CustomerHomeRemote`
```

This concludes the Exercises for Chapter 6. In Chapter 7 we will go through the xml descriptors for the EJBs, application descriptor and J2EE™ SDK descriptor.