DEVELOPING CMP 2.0 ENTITY BEANS

Topics in This Chapter

• Characteristics of CMP 2.0 Entity Beans
• Advantages of CMP Entity Beans over BMP Entity Beans
• CMP 2.0 Entity Bean Sample Application
Chapter 11

The CMP 2.0 entity bean represents a major upgrade from CMP 1.1 entity beans. In the previous chapter, we discussed the new features in CMP 2.0 entity beans, namely, abstract methods, abstract persistent schema, the EJB QL, and the respective responsibilities of bean developers, application assemblers, and deployers. To get a better understanding of how CMP 2.0 entity beans work in the real world, let’s apply that knowledge to implement a CMP 2.0 entity bean in this chapter.

This chapter discusses how to implement CMP 2.0 entity beans by reviewing

- the characteristics of CMP 2.0 entity beans
- when to use CMP 2.0 entity beans instead of BMP entity beans
- a step-by-step guide to implementing, packaging, and deploying CMP 2.0 entity beans

Characteristics of CMP 2.0 Entity Beans

CMP 2.0 entity beans have the following significant characteristics:

- CMP 2.0 entity beans support rich relationships between CMP 2.0 entity bean instances. Inter-entity bean relationship is characterized by its cardinality. CMP 2.0 entity beans support three types of cardinality: one-to-one, one-to-many, and many-to-many. This extensive
Developing EJB 2.0 Components

cardinality enables developers to model complex business relationships in applications.

- CMP 2.0 supports an abstract persistence schema that separates the client view of persistence from the implementation. The consequence is that developers can implement business logic based on an object model, without having to deal with the intricacies of the relational database model. During deployment, the abstract persistence schema model is mapped to the existing operational model. CMP 2.0 entity beans support EJB QL, a language for querying entity attributes and relationships between objects. At deployment time, the deployment tool converts EJB QL query statements to the query language of the underlying data source. CMP 2.0 entity beans use abstract methods to access container-managed fields and container-managed relationship fields.

- CMP 2.0 entity beans provide two new optional ejbHome and ejbSelect methods to perform global operations, thus providing developers with added flexibility to implement complex business logic.

- Bean developers don’t have to implement finder and ejbSelect methods; the deployment tool is responsible for generating the implementation classes during deployment.

- The container is responsible for automatically providing the inter-entity relationship referential integrity checks as well as implementing cascade delete. In CMP 2.0 relationships, cascade delete enables the container to delete dependent entity objects automatically when a primary entity object is deleted, all the while ensuring referential integrity in the relationship.

- CMP 2.0 beans must be declared an abstract class. During deployment, the vendor-provided deployment tool is responsible for subclassing the abstract bean class and generating concrete classes to support the container-managed persistence.

Advantages of CMP Entity Beans over BMP Entity Beans

Factors that influence a developer’s decision to use CMP entity beans as opposed to BMP entity beans include the availability of in-house expertise and the extent of a developer’s experience. There are several additional reasons to consider CMP over BMP:

- Time to market—With CMP entity beans, the developers write only the business logic and defer the persistence and relationship management logic to the deployment tool and the container, with the result that their applications contain fewer lines of code and take less time to develop.
Developing CMP 2.0 Entity Beans

With BMP entity beans, on the other hand, the developer is responsible for writing the persistence and relationship management logic in addition to the business logic.

- **Portability**—With BMP entity beans, the developer may write optimized SQL query statements and logic to manage persistence for a specific relational database. The hard-coded optimized SQL query statements and logic may not be portable to other relational databases. With CMP entity beans, the developer uses the abstract persistence schema to specify the CMP and CMR fields in the deployment descriptor. The vendor-provided deployment tool then generates the appropriate classes at deployment time, thus ensuring a high degree of portability regardless of the type of data source.

- **Flexibility**—With BMP entity beans, the developer must write the appropriate query statements to manage persistence based on the target data source. For example, the developer must write SQL statements for the relational database and OQL statements for the object database. As a result, third-party EJB providers must code and provide two sets of data access objects. The end users then must use the right combination of data access logic and query language for the query statements according to the target database. This adds unnecessary code management tasks for the user and the seller of BMP entity bean components. With CMP entity beans, on the other hand, the developer uses the abstract persistent schema to declare the CMP and CMR fields and then specifies the query using the EJB QL in the deployment descriptor. The deployment tool provides the flexibility to generate the SQL query for a relational database or the OQL query for an object database.

- **Improved performance**—To enable high performance with BMP entity beans, bean developers (the business domain experts) must also become database experts, as they must write optimized code for a particular vendor’s database. Obviously, database expertise is usually the domain of database administrators, not of bean developers. A higher level of data access code optimization also leads to reduced portability of the bean class. With CMP entity beans, the vendor-provided tool can read the deployment descriptor and potentially generate highly optimized code for the target data source. The degree of code optimization (and, therefore, of real-world CMP entity bean performance) will vary among the container providers. Optimization is a matter of simply converting EJB QL into native API calls of the particular target data source during deployment. Bean developers don’t have to learn any vendor-specific APIs to improve performance in CMP entity beans.

- **Referential integrity**—CMP entity beans inherit the rich relationship semantics, referential integrity, cardinality, relationship management, and
Developing EJB 2.0 Components

cascading delete that the EJB 2.0 container provides automatically. With BMP entity beans, on the other hand, the bean developer must provide referential integrity checks and relationship management when implementing inter-entity relationships—and that’s no trivial task.

• **Ease of maintenance**—With BMP entity beans, there’s more code to maintain—data access code in addition to business logic code results in software code maintenance challenges. CMP entity beans have only business logic code, as the deployment tools generate complicated data access and management code automatically during deployment.

Clearly, CMP entity beans have overwhelming advantages over BMP entity beans. They do have some drawbacks, however:

• **Debug difficulty**—Because the data access and management classes are generated by the vendor’s tool (meaning that the bean developers don’t have access to the source code), some bugs are harder to debug in CMP entity beans. Also, CMP entity bean implementation automatically inherits any bugs inherent in the persistence class generated by the deployment tool.

• **Reduced developer control**—The developer has complete control of the data access logic when using BMP entity beans and, consequently, more control over the entity bean’s persistence management. In some instances—such as when there’s a requirement to use vendor-specific, optimized query features like automatic primary key generators—the benefit of better data access control may be critical.

• **Higher learning curve**—Because most developers are already familiar with writing SQL queries, it’s easier to learn how to implement BMP entity beans. The CMP 2.0 abstract programming model is relatively more complex, and there’s a sharp learning curve initially. To implement CMP 2.0 entity beans, the bean developer must understand the abstract persistence schema model and abstract CMP and CMR methods.

**CMP 2.0 Entity Bean Sample Application**

The CMP 2.0 entity bean application that we describe in this chapter allows students to create their user IDs, add one or more addresses, and register for classes. It also allows the student to list the addresses and classes for which they are currently registered. The application consists of several JSP clients that invoke create and search methods on three CMP 2.0 entity beans: **StudentEJB**, **AddressEJB**, and **RosterEJB**.
Let’s examine the relationships among these three EJBs. A student can have zero or more addresses; as such, the student-address relationship is one-to-many and is a unidirectional relationship, navigating from student to address. When the student account is deleted, the address must also be deleted, requiring the use of a cascade delete option. The relationship between the student and the roster can be many-to-many (for example, a student can enroll in many classes, and a class can have many students); but, for simplicity, the relationship between student and roster is specified as one-to-many. There is no directional relationship between the address and the roster except via the student.

StudentEJB, AddressEJB, and RosterEJB are implemented as CMP 2.0 entity beans with local interfaces. The home and component interfaces of these EJBs are described and implemented in this chapter.

Steps necessary to implement CMP 2.0 entity beans are similar to the steps for implementing BMP entity beans; however, although coding for CMP entity beans is simpler, deployment is more complicated. The tasks involved in the development and deployment of the sample application include, in sequence:

1. Implement the CMP 2.0 address entity bean (AddressEJB).
2. Implement the CMP 2.0 roster entity bean (RosterEJB).
3. Implement the CMP 2.0 student entity bean (StudentEJB).
4. Implement several JSP clients to test the CMP 2.0 entity beans.
5. Package the CMP 2.0 entity beans as EJB components.
6. Package the JSP as a Web component.
7. Deploy the CMP 2.0 entity bean sample application.
8. Test the sample application.

We’ll conclude with a discussion of the deployment descriptor, which results from steps followed during the packaging of an application.

Step 1: Implementing the CMP 2.0 Address Entity Bean

First, implement the local interfaces—LocalAddress, LocalAddressHome, and the AddressEJB entity bean classes. The AddressEJB bean is used to create and manage address-related information.
Developing EJB 2.0 Components

Implementing the Local Home Interface: LocalAddressHome.java

The LocalAddressHome interface declares two methods: create() and the mandatory findByPrimaryKey(). Notice that both methods return the LocalAddress interface and must not throw RemoteExceptions, as shown in the following example:

```java
public interface LocalAddressHome extends EJBLocalHome {
    public LocalAddress create (String customerID, String addressID, String street, String city, String zip, String state)
        throws CreateException;
    public LocalAddress findByPrimaryKey(String addressID)
        throws FinderException;
}
```

Implementing the Local Component Interface: LocalAddress.java

The LocalAddress interface declares basic getter methods to retrieve the CMP address fields from the bean class:

```java
public interface LocalAddress extends EJBLocalObject {
    public String getAddressID();
    public String getStreet();
    public String getCity();
    public String getZip();
    public String getState();
}
```

Implementing the CMP 2.0 Entity Bean Class: AddressEJB.com

The CMP 2.0 entity bean class must be declared abstract; it extends the EntityBean interface. Notice that the abstract methods access the container-managed persistent fields such as addressID, street, city, zip code, and state:

```java
public abstract class AddressEJB implements EntityBean {
    private EntityContext context;
    //access methods for cmp fields

    public abstract String getAddressID(); //primary key
    public abstract void setAddressID(String id);

    public abstract String getStreet();
    public abstract void setStreet(String street);
```
public abstract String getCity();
public abstract void setCity(String city);

public abstract String getZip();
public abstract void setZip(String zip);

public abstract String getState();
public abstract void setState(String state);

The `ejbCreate()` method uses the abstract setter methods to set the persistent fields. Please refer to Chapter 9 for a detailed discussion and rules on writing `ejbCreate()` and `ejbPostCreate()` methods. The container automatically saves the fields in the database table. The `ejbCreate()` method returns a String primary key, as shown here:

```java
public String ejbCreate(String sid,String id,String street,String city,String zip,String state)
     throws CreateException {
    setAddressID(id);
    setStreet(street);
    setCity(city);
    setZip(zip);
    setState(state);
    return null;//explanation at the end of the chapter
}
```

Look at the `ejbPostCreate()` method; it looks up the student object and adds the roster object to the student list. This allows the StudentEJB instance to access AddressEJB.

```java
public void ejbPostCreate(String sid,String id,String street,String city,String zip,String state)
     throws CreateException {
    try {
        Context ic = new InitialContext();
        LocalStudentHome home = (LocalStudentHome)
            ic.lookup("java:comp/env/ejb/StudentRef");
        LocalStudent student = home.findByPrimaryKey(sid);
        student.addAddress((LocalAddress)context.getEJBLocalObject());
    } catch (Exception ex) {
        context.setRollbackOnly();
        ex.printStackTrace();
    }
}
```
Compiling `LocalAddressHome.java`, `LocalAddress.java`, and `AddressEJB.java`

In your Windows terminal, change directory to `APPHOME\chapter11\cmp` and run `compileAddress.bat` to generate the `LocalAddressHome.class`, `LocalAddress.class`, and `AddressEJB.class` in your directory.

Step 2: Implementing the CMP 2.0 Roster Entity Bean

The Roster entity bean, `RosterEJB`, creates a roster of students. It has a local home interface (`LocalRosterHome`) and a local component interface (`LocalRoster`).

Implementing the Local Home Interface: `LocalRosterHome.java`

`LocalRosterHome` declares the `create()`, `findByPrimaryKey()`, and additional finder methods. The `findByScheduleID()` method returns a collection based on the schedule ID; the `findByStudentID()` method returns a collection based on the student ID argument.

```java
public interface LocalRosterHome extends EJBLocalHome {
    public LocalRoster create(String rosterID, String scheduleID, String studentID) throws CreateException;
    public LocalRoster findByPrimaryKey(String rosterID) throws FinderException;
    public Collection findByScheduleID(String scheduleID) throws FinderException;
    public Collection findByStudentID(String studentID) throws FinderException;
}
```

Implementing the Local Component Interface: `LocalRoster.java`

The `LocalRoster` component declares three plain getter business methods that return persistent fields as shown next.

```java
public interface LocalRoster extends EJBLocalObject {
    public String fetchRosterID();
    public String fetchStudentID();
    public String fetchScheduleID();
}
```
Implementing the CMP 2.0 Entity Bean Class: RosterEJB.com

The code snippet that follows illustrates an abstract Roster entity bean class with several abstract accessor methods for CMP fields. Note that the business method, fetchRosterID(), depends on the abstract method getRosterID() to retrieve the CMP fields from the underlying database.

```java
public abstract class RosterEJB implements EntityBean {
    //accessor methods for cmp fields
    public abstract String getRosterID();
    public abstract void setRosterID(String rosterID);
    public abstract String getScheduleID();
    public abstract void setScheduleID(String scheduleID);
    public abstract String getStudentID();
    public abstract void setStudentID(String studentID);

    //business methods
    public String fetchRosterID () {
        return getRosterID();
    }
    public String fetchStudentID() {
        return getStudentID();
    }
    public String fetchScheduleID() {
        return getScheduleID();
    }

    The ejbCreate() method is shown next. It calls no data access calls but rather uses the setter methods to set the persistent fields.

    public String ejbCreate (String rosterID, String scheduleID, String studentID)
    throws CreateException {
        System.out.println("RosterEJB.ejbCreate...");
        setRosterID(rosterID);
        setScheduleID(scheduleID);
        setStudentID(studentID);

        return null;
    }
```
Developing EJB 2.0 Components

The `ejbPostCreate()` method performs the JNDI lookup for the Student bean reference. It then uses the component interface to add the instance to the student instance object so it can be accessed by the student bean instance.

```java
public void ejbPostCreate (String rosterID, String scheduleID, String studentID) throws CreateException {
    System.out.println("RosterEJB ejbPostCreate...rosterID = "+rosterID);
    try {
        Context ic = new InitialContext();
        LocalStudentHome home = (LocalStudentHome)ic.lookup("java:comp/env/ejb/StudentRef");
        LocalStudent student = home.findByPrimaryKey(sid);
        student.addRoster((LocalRoster)context.getEJBLocalObject());
    } catch (Exception ex) {
        context.setRollbackOnly();
        ex.printStackTrace();
    }
}
```

Compiling `LocalRosterHome.java`, `LocalRoster.java`, and `RosterEJB.java`

If you are not already in the APPHOME\chapter11\cmp directory, change to this directory and run the `compileRoster.bat` batch file to compile the Roster bean class. The batch file generates `LocalRosterHome.class`, `LocalRoster.class`, and `RosterEJB.class`.

Step 3: Implementing the CMP 2.0 Student Entity Bean

The student entity bean provides the logic to create a student entity. It accesses the address bean and roster bean instance to look up the address entity assigned to the student and the classes for which the student is registered.

Implementing the Local Home Interface: `LocalStudentHome.com`

The local home interface declares the `create()` method and several finder methods. The `create()` method creates the student entity; the finder methods allow clients to query by primary key, student’s first name, and last name.

```java
public interface LocalStudentHome extends EJBLocalHome {
    public LocalStudent create (String StudentID, String firstName, String lastName) throws CreateException;
    public Collection findByLastName (String lastName)
```
Implementing the Local Component Interface: LocalStudent.java

The local component interface declares the getter methods. In addition, several business methods [including getAddressList() and getRosterList()] return a collection of LocalAddress and LocalRoster objects to the client. The client uses getAddressList() and getRosterList() to retrieve multiple addresses and the classes for which the student is registered. The addAddress() and addRoster() methods add local components to an existing collection.

```java
public interface LocalStudent extends EJBLocalObject {
    public String getStudentID();
    public String getFirstName();
    public String getLastName();
    public ArrayList getAddressList();
    public ArrayList getRosterList();
    public void addAddress(LocalAddress address);
    public void addRoster(LocalRoster roster);
}
```

Implementing the CMP 2.0 Entity Bean Class: StudentEJB.com

The student entity bean class uses abstract methods to declare several CMP fields, including studentID, firstName, and lastName. The unidirectional, one-to-many relationship between the student bean and address bean is represented by the CMR field addresses defined by the getAddresses() and setAddresses() abstract methods. Similarly, the rosters represent the unidirectional one-to-many relationships between student and roster beans.

```java
public abstract class StudentEJB implements EntityBean {
    //access methods for cmp fields
    public abstract String getStudentID();    //primary key
    public abstract void setStudentID(String id);
    public abstract String getFirstName();
    public abstract void setFirstName(String firstName);
    public abstract String getLastName();
```
The StudentEJB class declares two CMR fields to define its one-to-many relationships with the AddressEJB and RosterEJB bean classes. The getAddresses() and getRosters() abstract methods return collections of local component objects of the respective bean type. The setAddresses() and setRosters() abstract methods take collection arguments of the local components of the respective bean type, as shown in the code fragment that follows.

```java
//abstract cmr field methods for address entity bean
public abstract Collection getAddresses();
public abstract void setAddresses(Collection addresses);

//abstract cmr field methods for roster entity bean
public abstract Collection getRosters();
public abstract void setRosters(Collection rosters);
```

The business method getAddressList() calls the CMR abstract method getAddresses(), which returns a collection. The getAddressList() method then extracts the local component interface LocalAdress, creates an array list using the Iterator, and returns it to the client. The client can then invoke business methods on individual local components. The addAddress() method is used by the client to add a local component interface, LocalAddress, to the collection of LocalAddress objects. The getRosterList() and addRoster() methods (not shown here) mirror the same functionality for the roster bean. When a client invokes getAddressList(), it receives a list of LocalAddress interfaces and uses it to invoke business methods.

```java
//business methods
public ArrayList getAddressList() {
    ArrayList list = new ArrayList();
    Iterator c = getAddresses().iterator();
    while (c.hasNext()) {
        list.add((LocalAddress)c.next());
    }
    return list;
}
public void addAddress (LocalAddress address) {
    getAddresses().add(address);
}
```

The ejbCreate() and ejbPostCreate() methods in the StudentEJB class are fairly simple. The ejbCreate() method invokes the abstract methods to set the id, firstName, and lastName abstract persistent fields. The ejbPostCreate() method is left empty.
public String ejbCreate (String id, String firstName, String lastName) throws CreateException {
    System.out.println(" -- StudentEJB - ejbCreate...");
    setStudentID(id);
    setFirstName(firstName);
    setLastName(lastName);

    return id;
}

public void ejbPostCreate (String id, String firstName, String lastName) throws CreateException {
    System.out.println("StudentEJB - ejbPostCreate(" + id + ", " + firstName + ", " + lastName + ")...\n    ");
}

Compiling LocalStudentHome.java, LocalStudent.java, and StudentEJB.java

If you are not already in the APPHOME\chapter11\cmp directory, change to this directory and run the compileStudent.bat file to compile the Student bean class. The batch file creates LocalStudentHome.class, LocalStudent.class, and StudentEJB.class.

Step 4: Implementing JSP Clients to Test CMP 2.0 Entity Beans

To test the entity bean application, use JSP clients. For each entity bean, there are two JSPs—one to create the data and the other to search for it. We’ve written six JSP files: createAddress.jsp and searchAddress.jsp for AddressEJB, createRoster.jsp and searchRoster.jsp for RosterEJB, and createStudent.jsp and searchStudent.jsp for StudentEJB. Because JSP and clients’ implementation aren’t the focus of this book, we’ll look at just one JSP client, SearchStudent.jsp, and briefly explain its logic.

The input form is embedded in the JSP file, as shown next. Embedding the form allows three search options—student ID, last name, or first name.

Search for a Student:
<form method="get" action="/cmp3/searchStudent.jsp">
  Search by
  <select name="searchCriteria">
    <option value="studentID" selected>Student ID
    <option value="lastName">Last Name
    <option value="firstName">First Name
</form>
Developing EJB 2.0 Components

Depending on the search criteria, SearchStudent.jsp performs a JNDI lookup for a student home object and then calls finder methods, two of which return a collection. It next goes through the array list; then, using the student and roster component interface, it extracts the classes in which the students are enrolled and the students’ addresses.

```java
InitialContext ic = new InitialContext();
Object obj = ic.lookup("java:comp/env/ejb/StudentRef");
LocalStudentHome home = (LocalStudentHome) obj;

Collection Students = new ArrayList();
if ("studentID".equals(criteria)) {
    try {
        LocalStudent student = home.findByPrimaryKey(text);
        Students.add(student);
    } catch (ObjectNotFoundException ex) {} 
} else if ("lastName".equals(criteria)) {
    Students = home.findByLastName(text);
} else if ("firstName".equals(criteria)) {
    Students = home.findByFirstName(text);
}
```

The following code fragment extracts the LocalStudent interface, invokes business methods, and displays the addresses and classes in an html table as follows:

```
  for (int i = 0; i < Students.size(); i++) {
    LocalStudent stud = (LocalStudent)((ArrayList)Students).get(i);
    String sid = (String) stud.getPrimaryKey();
    ArrayList rosterList = stud.getRosterList();
    %>
    <b> <%=stud.getFirstName()%>  <%=stud.getLastName()%> </b> is reg-
    istered in
    <%=stud.getRosterList().size()%> classes listed below:  <p>
    <table border=2>
    <tr><th>Roster ID</th> <th>Schedule ID</th></tr>
    <%}
    for (int j=0; j < rosterList.size(); j++)
    {
```
Step 5: Packaging the CMP 2.0 Entity Beans as an ejb-jar File

To package the sample application, we first need to create the application archive file that will hold the ejb-jar and Web war files. Let’s first create the application archive file CMP20App.ear:

1. Create an ear file that will include the EJB jar and Web war files. Start the Sun J2SDKEE1.3 RI and the deployment tool. Using the deployment tool, select File | New | Application. When a new application window pops up, click the Browse button to go to the APPHOME\chapter11 subdirectory. Enter the name of the ear file (CMP20App.ear) as shown in Figure 11-1. Click New Application and then OK to create this file.

2. Package the EJB component by creating one ejb-jar file, then package all three EJBs into one jar file.
Developing EJB 2.0 Components

3. Select File | New | Enterprise Bean. When the Enterprise Bean Wizard pops up, click Next. In the Wizard window, ensure that the Create New Jar radio button is selected and displays CMP20App. If not, use the pull-down menu to select it.

4. In the JAR Display Name area, enter **CMP20JAR** and click Edit to bring up the Edit Contents window. Using the pull-down menu, select the APPHOME\chapter11 directory and note the subdirectories (including cmp and Web). Select the cmp directory icon; it will expand and display the contents of the directory. To add classes to the jar file, select AddressEJB.class, LocalAddressHome.class, and LocalAddress.class. Click the Add button, as shown in Figure 11-2, and then click OK. The Edit Content window will close. Click Next.
5. In the Wizard window, enter the general information regarding the AddressEJB. First click the radio button to specify that this is an entity bean; then, using the pull-down menu under Enterprise Bean Class, select AddressEJB.class. Use the default AddressEJB for the Enterprise Bean Class Name. Under the Local Interfaces, select LocalAddressHome and LocalAddress for the local home and local interface (see Figure 11-3). Click Next.

6. Now, specify the CMP fields, abstract persistence schema, and the primary key. In the Wizard, select the Container Managed Persistence (2.0) radio button. In the Fields To Be Persisted box, select all the fields. In the Abstract Schema Name area, enter Address; specify that primary key as a Java String type under the Primary Key Class text box by entering java.lang.String. Under Primary Key Field Name, use the pull-down menu to select addressID as the primary key field name (see Figure 11-4). Click the Next button twice.
Developing EJB 2.0 Components

Figure 11-3 Adding the AddressEJB bean to the CMP20JAR

Figure 11-4 Specifying CMP fields in the abstract persistence schema
7. The AddressEJB bean references StudentEJB in the code—at runtime, AddressEJB needs to locate the StudentEJB instance. Under the Coded Name column, enter ejb/StudentRef; under Type and Interfaces, select Entity and Local, respectively. Under the Home Interface and Local/Remote Interface column, enter the full path of the LocalStudentHome and LocalStudent interfaces. Under Enterprise Bean Name, enter StudentEJB. In this case, don’t specify the full path name (see Figure 11-5). Click Finish.

8. The parameters for AddressEJB are specified; now, we’ll repeat the same process for RosterEJB and StudentEJB.

9. To add the RosterEJB bean to the existing CMP20JAR file, select it. Then, using the deployment tool, select File | New | Enterprise Bean. Click Next in the Wizard and verify that Add to the Existing Jar File is selected and that CMP20JAR is displayed. Click the Edit button. When the Edit Content window appears, go to the CMP subdirectory under chapter11. Add LocalRosterHome.class, LocalRoster.class, and RosterEJB.class to the jar file by clicking the Add button for each (see Figure 11-6). Click OK and then Next.

10. In the Wizard, select Entity as the Enterprise Bean Class, and select RosterEJB from the pull-down menu. Select LocalRosterHome and LocalRoster under the Local Interfaces section as home and local interfaces (see Figure 11-7). Click Next.
11. In the Wizard, select Container Managed Persistence (2.0). Select all the fields to be persisted, enter Roster as the Abstract Schema Name, and from the pull-down menu, select java.lang.String as the primary key class and rosterID as the primary key field (see Figure 11-8). Click Next twice.

12. Because the RosterEJB class references StudentEJB, specify the JNDI name and the local references. Enter ejb/StudentRef as the Coded Name; select Local as the Type and Local as the Interface. Use the fully qualified LocalStudentHome and LocalStudent names for the home and local interfaces. Enter StudentEJB as the Enterprise Bean Class name (see Figure 11-9). Click Finish.
11 • Developing CMP 2.0 Entity Beans

Figure 11-7 Specifying the RosterEJB bean class and local interfaces

Figure 11-8 Specifying CMP 2.0 type and CMP fields in the abstract persistence schema
Developing EJB 2.0 Components

13. Add the StudentEJB bean to the jar file. Using the deployment tool, select File | New | Enterprise Bean. When the Wizard pops up, click Next and then Edit. When the Edit Content window pops up, go to the APPHOME\chapter11\cmp directory and add LocalStudentHome.class, LocalStudent.class, and StudentEJB.class to the CMP20JAR file by clicking the Add button. Click OK and then Next (see Figure 11-10).

14. Select Entity. Using the pull-down menu, select StudentEJB, LocalStudentHome, and LocalStudent as the Bean Class types and Local Home and Local Interface respectively (see Figure 11-11). Click Next.

15. In the Wizard, select the Container Managed Persistence (2.0) radio button; select studentID, firstName, and lastName as the fields to be persisted. Enter Student as the abstract schema name; enter java.lang.String as the primary key class; and enter studentID as the primary field (see Figure 11-12).

16. Because the LocalStudentHome interface declares two finder methods, specify the EJB QL query. Select Finder/Select Methods and enter the SELECT DISTINCT OBJECT(s) FROM Student AS s WHERE s.firstName = ?1 statement for the findByFirstName finder method.

17. Repeat the same for the EJB QL statement. Change the s.firstName to s.lastName (see Figure 11-13). Click OK in the Finder/Select Method window.
18. Because the StudentEJB code doesn’t reference any objects, there’s no need to specify a JNDI lookup. Click Finish to package all three EJBs into one jar file.

19. Now, specify the relationships between the EJBs and generate the SQL statements. Select the CMP20JAR icon and select the Relationships tab. Click Add to display the Add Relationship window, in which you use the pull-down menu to specify the relationship between the StudentEJB and the AddressEJB (a one-to-many relationship). Under Multiplicity, select One to Many. Click the Description button. In the window that pops up, enter Student-has-many-Address. Click Apply, and then click OK.

20. Specify the relationship between these two beans. Under Enterprise Bean A, use the pull-down menu to select StudentEJB as the Enterprise Bean Name, addresses for Field Referencing Bean B, and the java.util.Collection as
Developing EJB 2.0 Components

Figure 11-11 Specifying the StudentEJB bean class and local interfaces

Figure 11-12 Specifying CMP fields in the abstract persistence schema
Developing CMP 2.0 Entity Beans

21. Next, specify the relationship between the StudentEJB and the RosterEJB. StudentEJB has a one-to-many relationship with RosterEJB; StudentEJB references RosterEJB with the CMR field rosters specified by the CMR abstract method, which returns a collection. This is specified in the relationship window (see Figure 11-15).
22. Now, generate the SQL statement. Select the StudentEJB icon, select the Entity tab, and then select Deployment Settings to display the Deployment Settings window. Provide the database setting information and then click the Database Setting button to display another Database Setting window. Enter `jdbc/Cloudscape` under Database JNDI Name; enter `j2ee` as the user name and password; and then click OK (see Figure 11-16).

23. Next, click the Create Table on Deploy and Delete Table on Undeploy buttons; then, select Generate Default SQL. If the EJB QL query was correct, the SQL Generation Complete window pops up. On the deploytool window terminal, the generated SQL statement for both finder methods should appear. Click OK, then click the OK button in the deploytool window.
24. The LocalRosterHome interface also declared two finder methods—findByScheduleID and findByStudentID. Specify the EJB QL for these finder methods. (This could have been done when packaging RosterEJB, but we deferred this step to show how it can also be done later.) Select Finder/Select Methods to display the EJB QL window. Select the findByScheduleID(...) method; then, in the EJB-QL Query for FindByScheduleID window, enter `SELECT DISTINCT OBJECTR FROM Roster AS r WHERE r.scheduleID = ?1`.

25. Select the findByStudentID(...) method under Method. Enter the query statement from the previous step, but change `r.studentID` to `r.scheduleID` (see Figure 11-17). Click OK; then select Deployment Setting. When the Deployment Setting window appears, check Create Table on Deploy and Delete on Undeploy. Then select Generate Default SQL to generate the SQL
Developing EJB 2.0 Components

Figure 11-16 Specifying database deployment settings

statement for the finder methods. If the EJB QL was entered correctly, the SQL Generation Complete window should pop up. Click OK, and the window will disappear. Then click OK in the Deployment Setting window.

26. Select the AddressEJB icon and then the Entity tab. Select Deployment Setting to display the Deployment Setting window. Because the LocalAddressHome interface didn’t declare any finder methods, don’t specify any EJB QL query. Check both Create Table on Deploy and Delete Table on Undeploy; then, select Generate Default SQL. The SQL Generation Complete confirmation window should pop up. Click OK and then OK again in the Deployment Setting window.
Step 6: Packaging the JSP As a Web Component

We’ve just completed the packaging of the EJB component. Now, let’s package the Web components, consisting of JSP pages, into the CMP20WAR file.

1. First, package the JSP clients for AddressEJB. In the deployment tool, select the File | New | Web component, which brings up a Web Component Wizard. Click Next. In the Wizard, select the Create New WAR File in Application button to display CMP20App. In the WAR Display Name area, enter CMP20WAR and then click Edit. The Edit Content window will pop up. Go to the APPHOME\chapter11\web directory, and add index.html and createStudent.jsp to the WAR file (see Figure 11-18). Click OK, and then click Next.
2. Next, select the JSP radio button to indicate the type of Web component, and select Next from the pull-down menu. Select createStudent.jsp as the JSP Filename; take all the default values (see Figure 11-19). Click Next twice.

3. Now, specify the component alias name. Click Add and enter createStudent.jsp as shown in Figure 11-20. Click Next four times.

4. Because the createStudent.jsp client references StudentEJB, specify EJB references. Click Add and then enter ejb/StudentRef as the Coded Name. Use the pull-down menu to select Entity as the Type and local as the Interface. Enter LocalStudentHome and LocalStudent as the Home and Local/Remote Interfaces using the full path name (for example, j2eobootcamp.developingEJB.chapter11.cmp). Then, as the Enterprise Bean Name, enter StudentEJB (see Figure 11-21). Click Finish.
Figure 11-19  Specifying the createStudent.jsp type for the Web component

Figure 11-20  Specifying the JSP alias name
Developing EJB 2.0 Components

5. Now, create the searchStudent component. Select the File | New | Web component, and click Next. To package all the JSP files into one war file, select Add to Existing WAR File and use the pull-down menu to select CMP20WAR. Click Edit.

6. In the Edit Content window, go to the APPHOME\chapter11\cmp directory and add searchStudent.jsp (see Figure 11-22). Click OK; then Next, then JSP and Next again. Under JSP Filename, select searchStudent.jsp. Accept the default values (see Figure 11-19)—note that the name should say searchStudent). Click Next twice; click Add and enter searchStudent.jsp (see Figure 11-20)—note that the name should say searchStudent.jsp). Click Next four times. In the EJB reference window, enter ejb/StudentRef, Entity, Local, LocalStudentHome, and LocalStudent (see Figure 11-21). Click Finish.

7. The steps necessary to package createAddress.jsp is the same as the steps we used to package searchStudent.jsp. To package the createAddress.jsp file, follow the sequence of steps we outlined for packaging the searchStudent.jsp file. Now add createAddress.jsp (see Figure 11-18) to the CMP20WAR file; specify it as a JSP component, and specify the JSP filename (see Figure 11-19). Use createAddress.jsp as the alias (see Figure 11-20). Then, use ejb/AddressRef, entity, local, LocalAddressHome, LocalAddress, and AddressHome in the EJB reference (see Figure 11-21) and click Finish.
8. Repeat the same sequence of steps to package searchAddress.jsp, except in this case, replace createAddress.jsp with searchAddress.jsp and use the EJB references (ejb/AddressRef, entity, local, LocalAddressHome, LocalAddress, and AddressEJB) as in the previous steps. Click Finish.

9. Repeat the same series of steps to package createRoster and searchRoster. Replace the searchAddress.jsp with searchRoster.jsp and specify ejb/RosterRef, entity, local, LocalRosterHome, LocalRoster, and RosterEJB in the EJB reference window. Repeat the steps for searchRoster.jsp.

10. After you package both the EJB and JSP components, all six JSP components should be included under the CMP20WAR icon, and three entity beans should be included under the CMP20JAR icon. Select the CMP20App icon
Developing EJB 2.0 Components

and then select the JNDI Name tab. Enter **MyStudent**, **MyAddress**, and **MyRoster** under the JNDI Name column (see Figure 11-23).

Step 7: Deploying the CMP 2.0 Entity Bean Sample Application

We’re ready to deploy the sample application, now that we’ve packaged the EJB and Web components into a deployable CMP20App.ear file.

1. Select the CMP20App icon and select Tools | Deploy to bring up the Deploy window. Select CMP20App under Object to Deploy; enter **localhost** under Target Server; and check Save Object Before Deploying (see Figure 11-24). Click Next.

2. Verify that the JNDI names are correctly set, as shown in Figure 11-25. Click Next.
Under the ContextRoot column in the pop-up window, enter /cmp3 and then click Finish. The Deployment Progress window message appears. If deployment is successful, the message changes to Deployment of CMP20App is Complete. Click OK. (See the Appendix to this book for a description of possible common errors and how to fix them.)
Developing EJB 2.0 Components

Step 8: Testing the Sample Application

Now you’re ready to test the CMP 20 sample application by creating a student account and several addresses, enrolling the student in some courses, then searching for the student and listing the addresses and courses for which that student is registered.

1. Open a browser and enter http://localhost:8000/cmp3, as shown in Figure 11-26.
2. Create a student account. Select the Create New Student link, and the form in Figure 11-27 will appear. Click Submit to create the student account.

![Sample application start page](image)

**Figure 11-26** Sample application start page

![Creating a student entity](image)

**Figure 11-27** Creating a student entity
3. Create two addresses for the student, John Doe. Select the Create New Address link, and in the address create form, fill out the fields with jdoe as the customer ID, 100 as the address ID, 1 Market Street as the street address, San Francisco for the city name, CA for state, and 93456 as the zip code (see Figure 11-28). Click Submit to create the address.

4. Add another address with the data jdoe, 101, 99 Lake View Ave., Lake Tahoe, CA, 95324 for customer ID, address ID, address, city, state, and zip, respectively. Select the HOME link to return to the start page.

5. Register the student, John Doe, in couple of J2EE, EJB, and Java classes. Select Create New Roster, and then enter 1001 as the roster ID, J2EE101 as the schedule ID, and jdoe as the student ID (see Figure 11-29). Register for the EJB and Java classes—enter 1002, EJB-101, jdoe and 1003, Java-200, jdoe.

6. Search for student jdoe, then list all his associated addresses and the classes for which he is currently registered. Select the Search For Student link and enter jdoe. The results are shown in Figure 11-30.

A Discussion of the Deployment Descriptor

Let’s choose one EJB, StudentEJB, and look at the pertinent elements from the deployment descriptor file (ejb-jar.xml) that will help to elucidate the key aspects of the abstract persistent schema and the abstract persistent methods of CMP 2.0
Developing EJB 2.0 Components

Figure 11-29 Registering a student in several classes

Figure 11-30 Result of search
entity beans. First, we’ll review the structural information of StudentEJB that’s familiar to us, namely, the <local-home>, <local>, and <ejb-class> elements specify the LocalStudentHome and LocalStudent interfaces and the StudentEJB bean class. The <persistent-type> element specifies that the persistence is managed by the container, and the <cmp-version> element specifies that StudentEJB implements EJB 2.0 persistence. The primary key class is of type java.lang.String, as indicated by the <prim-key-class> element, and StudentEJB does not support callbacks, as specified by the <reentrant> element.

Now, let’s look at the abstract persistent schema. In the StudentEJB class, we declared several CMP abstract methods, such as set/getStudentID(), set/getFirstName(), and set/getLastName(). During the deployment of StudentEJB, we specified Student as the name of the abstract persistent schema and the EJB QL for finder methods. All of that information is saved in the deployment descriptor file, as shown in the code fragment that follows. Notice that the <abstract-schema-name> element specifies Student as the name of the schema, followed by the <cmp-field> element that encloses the optional <description> and mandatory <field-name> elements. The StudentID from the getStudentID() abstract method maps to the <field-name>studentID</field-name>, but notice that the first letter of StudentID must be lowercase. This rule applies to both the CMP and the CMR fields. The <primaryKey-field> element is used to specify the primary key for the entity bean.

Now, let’s look at the abstract persistent schema. In the StudentEJB class, we declared several CMP abstract methods, such as set/getStudentID(), set/getFirstName(), and set/getLastName(). During the deployment of StudentEJB, we specified Student as the name of the abstract persistent schema and the EJB QL for finder methods. All of that information is saved in the deployment descriptor file, as shown in the code fragment that follows. Notice that the <abstract-schema-name> element specifies Student as the name of the schema, followed by the <cmp-field> element that encloses the optional <description> and mandatory <field-name> elements. The StudentID from the getStudentID() abstract method maps to the <field-name>studentID</field-name>, but notice that the first letter of StudentID must be lowercase. This rule applies to both the CMP and the CMR fields. The <primaryKey-field> element is used to specify the primary key for the entity bean.
Developing EJB 2.0 Components

Notice that we declared three finder methods in the `LocalStudentHome` interface but didn’t have to implement them in the `StudentEJB` bean class. In CMP 2.0, the container is responsible for implementing the finder methods, so the finder method is mapped in the deployment descriptor with the `<query>` element. The `<query>` element encloses several elements—the `<query-method>` element that includes the method name as enclosed by `<method-name>`, and the argument-type, enclosed by the `<method-param>` element. If there were more than one argument, there would be a matching number of `<method-param>` elements. The finder method, `findByFirstName`, is mapped to the EJB QL query statement by the `<ejb-ql>` element that encloses the EJB QL statement specified during the deployment. With the exception of the `findByPrimaryKey()` method, for every other finder method declared in the home interface, the container will define `<query>` elements in the deployment descriptor.

```<query>
  <description></description>
  <query-method>
    <method-name>findByFirstName</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </query-method>
  <ejb-ql>SELECT DISTINCT OBJECT(s) FROM Student AS s WHERE s.firstName = ?1</ejb-ql>
</query>```
The relationship information is enclosed within the `<relationship>` element. During the deployment phase, we specified that the `StudentEJB` and `AddressEJB` bean instances had a one-to-many relationship; this is specified in the deployment descriptor by the `<multiplicity>` element of `One` for `StudentEJB` and of `Many` for `AddressEJB`. The `StudentEJB` bean class declared two CMR abstract methods, and the `getAddresses()` abstract method returned `java.util.Collection`. During deployment, we specified that `StudentEJB` has a one-to-many relationship with `AddressEJB` and that `StudentEJB` references `AddressEJB` with CMR fields `addresses`, which is a type of collection as shown in Figure 11-14. The `<cmr-field>` element uses two subelements to specify the CMR field; the `<cmr-field-name>` element specifies the CMR field and `addresses`; and the `<cmr-field-type>` element specifies the `Collection` type, as shown in the following code fragment. The `<cascade-delete>` element reflects your selection of the Delete When Bean A is Deleted option during deployment. Also notice that because `AddressEJB` declares no CMR abstract methods, there are no `<cmr-field>` elements declared in the `AddressEJB` section of the deployment descriptor.

```xml
<relationships>
    <ejb-relation>
        <ejb-relation-name>Student-has-many-Addresses</ejb-relation-name>
        <ejb-relationship-role>
            <ejb-relationship-role-name>StudentEJB</ejb-relationship-role-name>
        </ejb-relationship-role>
        <multiplicity>One</multiplicity>
        <relationship-role-source>
            <ejb-name>StudentEJB</ejb-name>
        </relationship-role-source>
        <cmr-field>
            <cmr-field-name>addresses</cmr-field-name>
            <cmr-field-type>java.util.Collection</cmr-field-type>
        </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role>
        <ejb-relationship-role-name>AddressEJB</ejb-relationship-role-name>
    </ejb-relationship-role>
</relationships>
```
Developing EJB 2.0 Components

Why return null in ejbCreate() method?

In EJB 1.0, the EJBCreate() method in CMP entity beans declared the returned type void but in EJB 1.1 this method was changed to return primary key types with an actual return value of null. This change was made to facilitate using BMP entity beans to extend CMP entity beans. In essence, EJB 1.1 enabled vendors to support container-managed persistence by extending CMP entity beans with BMP entity beans. This solution worked in EJB 1.1, but was not applicable to the complexity of CMP 2.0 entity beans. Nevertheless, it provides backward compatibility and enables subclassing of bean managed persistence. In general, avoid it in EJB 2.0 unless backward compatibility is one of your concerns.

Summary

In this chapter we discussed the chief characteristics of CMP 2.0 entity beans, such as their ability to form complex relationships between entity beans, enforcement of their referential integrity, and support for the abstract persistence schema and the standard EJB Query Language. We also discussed reasons why CMP 2.0 entity beans are preferred over BMP entity beans, including enhanced portability, flexibility with persistence implementation, better performance, and reduced time-to-market. We spent the major portion of the chapter discussing the implementation of the StudentEJB, RosterEJB, and AddressEJB CMP 2.0 entity beans, and we showed how to deploy and run the sample application. We’ll examine the Java Message Service (JMS) in the next chapter.