Exercise 8:

Creating a CMP Entity EJB™

Case Study: Creating a CMP Entity EJB

Estimated completion time: 1 hour 30 minutes.

In this exercise, you create a CMP entity EJB™ that functions as a sequence number generator. The `Sequencer` CMP entity EJB maintains a table of sequence numbers used as primary keys when adding a row to a database table. For example, when a new account is added to the `J2EEACCOUNT` table, a unique account number is needed as a primary key to differentiate the new row from other rows in the table. Before adding the row, a unique key is obtained from the `Sequencer` EJB. The `Sequencer` EJB maintains the sequence values in a database table. The SQL code used to generate the sequencer table is shown below. Each row in the `J2EESEQUENCER` table corresponds to a database table requiring primary key generation. In the banking application, both the `J2EEACCOUNT` and `J2EEAPPUSER` tables rely on the sequence generator to create a primary key when adding a new row.

```
CREATE TABLE J2EESEQUENCER(
      SEQUENCETYPE   VARCHAR(20)
         CONSTRAINT PK_J2EESEQUENCER PRIMARY KEY,
      SEQUENCEVALUE  INT);

INSERT INTO J2EESEQUENCER VALUES ('ACCOUNT', 103);
INSERT INTO J2EESEQUENCER VALUES ('APPUSER', 35);
```

As illustrated in Figure 1, when the `Account` EJB needs to add a new row to the `J2EEACCOUNT` table, it requests a new sequence value from the `Sequencer` EJB using the `getNextSequenceValue` method. The `Sequencer` EJB generates a new sequence value by locating the appropriate row in the `sequencer` table based on the `sequenceType`, "ACCOUNT" in this example, retrieving the current value from the `J2EESEQUENCER` table, incrementing the sequence value by one, returning the incremented value to the calling method as the new sequence value, and storing the new value back to the `J2EESEQUENCER` table.

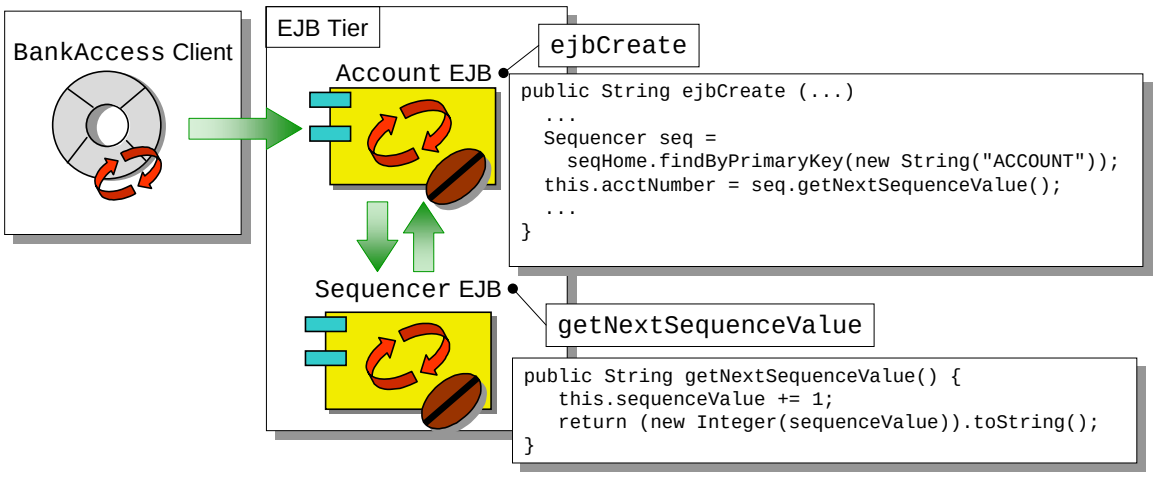

Figure 1 - Retrieving a Unique Primary Key

Note that using a CMP entity EJB as a sequence number generator is only one of the many ways to generate unique numeric primary keys.

There are four sections to this lab exercise. In the first section, you create the `Sequencer` CMP entity EJB interfaces and bean class. In Section 2, you modify the `Account` EJB to obtain a unique account number from the `Sequencer` EJB when adding a new account to the `J2EEACCOUNT` table. In Section 3, you code the `BankAccess` test client to invoke the new version of the `Account` EJB's `create` method. Finally, in Section 4, you deploy and test the application components.

After completing this lab, you will be able to:

- Create a simple CMP entity EJB
- Explain the difference between CMP and BMP entity EJBs
- Describe the steps required to deploy a CMP entity EJB into the J2EE runtime environment

Discussion Topics:

- What other mechanisms could you use to generate a unique primary key when adding a row to a database table?
- Discuss when using an entity EJB to generate unique primary keys might be inappropriate.

Important Terms and Concepts

Refer to the JDK Javadoc or the lab Appendix for information on the important classes and methods used in this exercise including parameters and return values.

java.rmi

```
public class RemoteException extends IOException
```

java.util

```
public interface Collection
public class Vector extends AbstractList implements List, Cloneable,
                                    Serializable
   public void addElement(Object obj)
public interface Iterator
   public boolean hasNext()
```

javax.naming

```
public interface Context
public class InitialContext extends java.lang.Object implements
                                    Context
   java.lang.Object lookup(java.lang.String name)
```

javax.rmi

```
public class PortableRemoteObject extends Object
   public static Object narrow(Object narrowFrom, Class narrowTo)
         throws ClassCastException
```

Setup Instructions

The `BankAccess` class serves as the test client for this lab. Use the "Guidelines for Creating a CMP Entity EJB" or the accompanying course material as a resource when completing this exercise. You can view an example of the completed exercise in the "Solution Code" section of this lab module.

Section 1

Define the `Sequencer` CMP Entity EJB's Home and Remote Interface

Use the "Guidelines for Creating a CMP Entity EJB" or the accompanying course material as a resource when completing this exercise.

1. Create a home interface for the `Sequencer` EJB in the `J2EEApp` package. The `Sequencer` EJB does not require the ability to add a row to the `Sequencer` data store, and therefore, does not define a creator method. The `Sequencer` EJB provides all of its searching capabilities based on the mandatory `findByPrimaryKey` method.
   - Name: `SequencerHome`
   - Creator/finder methods:
     - `findByPrimaryKey(String key)` – Locates a row in the `J2EESEQUENCER` table based on an identifying string
2. Create a remote interface for the `Sequencer` EJB in the `J2EEApp` package.
   - Name: `Sequencer`
   - Business methods:
     - `String getNextSequenceValue` – Returns a unique sequence value
3. Create a skeleton bean class for the `Sequencer` EJB that implements the methods defined on the EJB's home and remote interfaces, as well as any other methods required by the EJB container. Also add any fields required by the bean.
   - Name: `SequencerEJB`
   - Private fields:
     - `EntityContext context`
   - Persistent fields:
     - `String sequenceType`
     - `int sequenceValue`

Verify Your Work

4. Compare your work to the solution code provided for this lab exercise. Compile the `Sequencer` EJB's home and remote interfaces. Correct any errors or omissions before continuing.

Discussion Topics:

- How would implementing a BMP `Sequencer` EJB's home and remote interfaces differ from the CMP implementation used in this lab?

Code the `Sequencer` CMP Entity EJB's Bean Class

Follow the steps listed below to finish implementing the `Sequencer` EJB's bean class.

5. Modify the `Sequencer` EJB's bean class as follows:
   - Update `setEntityContext` to store the passed parameter to the `context` field.
   - Update `ejbActivate` to assign a value to the `sequenceType` field when the bean instance is activated.
   - Update `ejbPassivate` to reset the bean's persistent fields when the bean instance is passivated.
   - Code `getNextSequenceValue` to increment the current value of `sequenceValue` and return the new value as a `String`.

Compile

6. Build the files contained in the `J2EEApp` package. The entire package should compile without errors. Fix any errors before continuing with this exercise. Verify your work against the solution code provided for this module, if necessary.

Discussion Topics:

- How would implementing a BMP `Sequencer` EJB's bean class differ from the CMP implementation used in this lab?

Section 2

Modify the `Account` BMP Entity EJB

In this section, you add a new creator method to the `Account` EJB that utilizes the `Sequencer` EJB to generate a unique account number for the new table row instead of relying on a value passed to the method as a parameter. Use the "Guidelines for Creating a BMP Entity EJB" or the accompanying course material as a resource when completing this exercise.

1. Modify the `Account` entity EJB's home interface as follows:
   - Creator/finder methods:
     - `create(String customerID, String description, double balance)`
2. Add a method to the `Account` EJB's bean class to implement the new three parameter creator method. This method is similar to the existing creator method differing only in the mechanism used to obtain the `acctNumber` for the new account. Code the method to:
   - Use JNDI to locate the `Sequencer`'s home interface
   - Locate the row in the `J2EESEQUENCER` table that maintains the sequence values for the `J2EEACCOUNT` table. The `SEQUENCETYPE` for the `J2EEACCOUNT` table is "ACCOUNT"
   - Retrieve a new sequence value from the `Sequencer` EJB
   - Add a new row to the `J2EEACCOUNT` table and initialize the data elements for the new row using the sequence value and the passed parameters.
3. Add an `ejbPostCreate` method to compliment the new three parameter `ejbCreate` method. The method requires no code in the method body.

Verify Your Work

4. Compare your work to the solution code provided for this lab exercise. Compile the `Account` EJB's components. Correct any errors or omissions before continuing.

Discussion Topics:

- How does the container determine which of the `Account` EJB's two creator methods to invoke when a client application requests that a new row be added to the `J2EEACCOUNT` data store?

Section 3

Modify the `BankAccess` Test Client

The `BankAccess` class serves as the test client for this exercise.

1. Modify the `BankAccess`'s `main` method to call the `Account` EJB's new creator method and add a row to the `J2EEACCOUNT` table.
   - Add a row the the `J2EEACCOUNT` table using the new creator method, and print out the contents of the newly added row.
   - Delete the newly added row from the `J2EEACCOUNT` table.

Compile

2. Build all of the files in the `J2EEApp` package. Correct any errors before continuing with this exercise.

Discussion Topics:

- What method does an EJB client use to delete a row from the application data store?

Section 4

Deploy and Test

In this section, you deploy and test the application EJBs. Consult the "Assembly and Deployment Guidelines" for information on deploying EJB's into the J2EE container used for this course. Use the "IDE Configuration Guidelines" to assist you in modifying the IDE runtime configuration to access components running in the J2EE container.

1. Using the J2EE assembly/deployment tool provided for this course, assemble and deploy the application components into the J2EE runtime environment. Use "accountServer", "customerServer", "bankmgrServer", "sequencerServer", and "jdbc/BankMgrDB" as the JNDI name for the `Account`, `Customer`, `BankMgr`, and `Sequencer` EJBs and the database factory resource respectively. To assemble and deploy the application components using the J2EE Reference Implementation deployment tool, you must:
   - Start the J2EE runtime, cloudscape database, and deployment tool.
   - Use the deployment tool to:
     - Undeploy the `BankApp` application.
     - Update the application files for `BankApp`.
     - Update the method transaction attributes for the `AccountBean` and set the transaction attribute for the new creator method to "required".
     - Add the `Sequencer` entity bean to the application:
       - **New Enterprise Bean Wizard - EJB JAR**
         - Enterprise Bean Will Go In: `BankJAR`
         - Contents - Add:
           - Choose the system workspace as the root directory. For example, `C:\<IDEHome>\Development`.
           - Open the `J2EEApp` package.
           - Select and add `Sequencer.class`, `Sequencer EJB.class`, and `Sequencer Home.class` from the `J2EEApp` package.
       - **New Enterprise Bean Wizard - General**
         - Enterprise Bean Class: `J2EEApp.SequencerEJB`
         - Home Interface: `J2EEApp.SequencerHome`
         - Remote Interface: `J2EEApp.Sequencer`
         - Display Name: `SequencerBean`
         - Bean Type: Entity
       - **New Enterprise Bean Wizard - Entity Settings**
         - Container-Managed Persistence: Checked
         - Check fields: `sequenceType` and `sequenceValue`
         - Primary Key Class: `java.lang.String`
         - Primary Key Field Name: `sequenceType`

- **New Enterprise Bean Wizard - Transaction Management**
  - Set to “Required” for all methods.

- Modify the `SequencerBean`’s Deployment Settings found on the Entity Tab:
  - Database Settings:
    - Database JNDI Name: `jdbc/Cloudscape`
  - Database Table:
    - Create Table on Deploy: Unchecked
    - Delete Table on Undeploy: Unchecked
  - SQL for Database Access:
    - Click Generate SQL Now
    - Update the SQL code for the EJB methods as follows:
      - `ejbStore`:
        - `UPDATE J2EESEQUENCER SET SEQUENCEVALUE = ? WHERE SEQUENCETYPE = ?`
      - `ejbCreate`:
        - `INSERT INTO J2EESEQUENCER (SEQUENCETYPE, SEQUENCEVALUE) VALUES (?,?)`
      - `ejbRemove`:
        - `DELETE FROM J2EESEQUENCER WHERE SEQUENCETYPE = ?`
      - `findByPrimaryKey`:
        - `SELECT SEQUENCETYPE FROM J2EESEQUENCER WHERE SEQUENCETYPE = ?`
      - `ejbLoad`:
        - `SELECT SEQUENCEVALUE FROM J2EESEQUENCER WHERE SEQUENCETYPE = ?`
      - Table Create:
        - `CREATE TABLE J2EESEQUENCER  (SEQUENCETYPE VARCHAR(6) SEQUENCEVALUE INTEGER NOT NULL, CONSTRAINT "pk_SequencerEJBTable" PRIMARY KEY (SEQUENCETYPE))`
      - Table Delete:
        - `DROP TABLE J2EESEQUENCER`

- Deploy the application:
  - Select the option to Return Client Jar, and store the client `.jar` file in the *<J2EEAppHome>* directory.
  - Use “accountServer”, “customerServer”, “sequencerServer”, and “bankmgrServer” as the JNDI names for the `Account`, `Customer`, and `Sequencer` entity and `BankMgr` session EJBs, respectively.
  - Map the resource factory reference (`jdbc/BankMgrDB`) for the `BankMgr`, `Account`, `Sequencer`, and `Customer` EJBs to the name of the data source as registered with the J2EE runtime; for example, use “jdbc/Cloudscape” if you are using the Cloudscape database supplied with the J2EE Reference Implementation.

2. Configure the test client with the path to the client `.jar` file, and execute the test client. View the results in the IDE’s run console. Output generated by `println` statements in the EJB can be seen in the command window used to start the J2EE runtime system.

Solution Code: Section 1

```
// SequencerHome.java

package J2EEApp;

import java.rmi.RemoteException;
import javax.ejb.*;

public interface SequencerHome extends EJBHome {

   public Sequencer findByPrimaryKey(String key)
      throws RemoteException, FinderException;
}

************************************************************

// Sequencer.java

package J2EEApp;

import java.rmi.RemoteException;
import javax.ejb.*;

public interface Sequencer extends EJBObject {

   public String getNextSequenceValue() throws RemoteException;
}

************************************************************

// SequencerEJB.java

package J2EEApp;

import javax.ejb.*;

public class SequencerEJB implements EntityBean {

   public String sequenceType;
   public int sequenceValue;
   private EntityContext context;

   public SequencerEJB() { }

   public void setEntityContext(EntityContext ec) {
      this.context = ec;
   }

   public void ejbActivate() {
      sequenceType = (String)context.getPrimaryKey();
   }

   public void ejbPassivate() {
      sequenceType = null;
      sequenceValue = 0;
   }
```

```
   public void unsetEntityContext() { }
   public void ejbRemove() throws RemoveException { }
   public void ejbLoad() { }
   public void ejbStore() { }

   // business methods
   public String getNextSequenceValue() {
      System.out.println
         ("Entering SequencerEJB.getNextSequenceValue()");
      System.out.println("sequenceValue is: " + this.sequenceValue);
      this.sequenceValue += 1;
      System.out.println
         ("sequenceValue set to: " + this.sequenceValue);
      System.out.println
         ("Leaving SequencerEJB.getNextSequenceValue()");
      return (new Integer(sequenceValue)).toString();
   }
}

*************************************************************
```

Solution Code: Section 2

```
// AccountHome.java

package J2EEApp;

import java.rmi.RemoteException;
import javax.ejb.*;

public interface AccountHome extends EJBHome {

   public Account create(String acctNumber, String customerID,
                         String description, double balance)
      throws RemoteException, DuplicateKeyException, CreateException;

   public Account create(String customerID, String description,
                         double balance)
      throws RemoteException, DuplicateKeyException, CreateException;

   public Account findByPrimaryKey(String key)
      throws RemoteException, FinderException;

   public java.util.Collection findByCustomerID(String customerID)
      throws RemoteException, FinderException;
}

************************************************************

// AccountEJB.java

package J2EEApp;

import javax.ejb.*;
import java.sql.*;
import javax.naming.*;
import javax.sql.*;
import java.util.*;

public class AccountEJB implements EntityBean {

   private String acctNumber;
   private String customerID;
   private String description;
   private double balance;

   private EntityContext context;

   public AccountEJB() { }

   public String ejbCreate (String customerID, String description,
                            double balance)
         throws DuplicateKeyException,CreateException {
      System.out.println
         ("Entering AccountEJB.ejbCreate() w/sequencer");
      try {
         Context c = new InitialContext();
         Object result = c.lookup("sequencerServer");
         SequencerHome seqHome =
```

```
            (SequencerHome)javax.rmi.PortableRemoteObject.
               narrow(result,SequencerHome.class);
         Sequencer seq =
            seqHome.findByPrimaryKey(new String("ACCOUNT"));
         this.acctNumber = seq.getNextSequenceValue();
      } catch (Exception e) {
         System.out.println
            ("Error obtaiing seq# in AccountEJB.ejbCreate()");
         throw new CreateException ("Exception in create:" + e);
      }
      System.out.println("New sequence number is: " + this.acctNumber);
      this.customerID = customerID;
      this.description = description;
      this.balance = balance;
      AccountDAO ADAO = new AccountDAO();
      ADAO.setAcctNumber(this.acctNumber);
      ADAO.setCustomerID(customerID);
      ADAO.setDescription(description);
      ADAO.setBalance(balance);
      try{
         Connection dbConnection = getDBConnection();
         ADAO.create(dbConnection);
         dbConnection.close();
      } catch (java.sql.SQLException se) {
         throw new CreateException ("SQL Exception in create:" + se);
      }
      System.out.println("Leaving AccountEJB.ejbCreate() w/sequencer");
      return (acctNumber);
   }

   public String ejbCreate (String acctNumber, String customerID,
                            String description, double balance)
      throws DuplicateKeyException,CreateException { ... }

   public void setEntityContext(EntityContext ec) { ... }

   public void unsetEntityContext() { ... }

   public void ejbRemove() throws RemoveException { ... }

   public void ejbLoad() { ... }

   public void ejbStore() { ... }

   public void ejbActivate() { ... }

   public void ejbPassivate() { ... }

   public String ejbFindByPrimaryKey(String key)
      throws FinderException { ... }

   public Collection ejbFindByCustomerID(String customerID)
      throws FinderException { ... }

   public void ejbPostCreate(String acctNumber, String customerID,
                             String description, double balance)
      throws DuplicateKeyException,CreateException {
   }
```

```
   public void ejbPostCreate(String customerID, String description,
                             double balance)
      throws DuplicateKeyException,CreateException {
   }

   // business methods
   public AccountData getAccountData() { ... }

   // utility methods
   private Connection getDBConnection()
      throws SQLException { ... }
}
```

****

Solution Code: Section 3

// BankAccess.java

```
package J2EEApp;

import javax.naming.*;

public class BankAccess extends Object {

   private BankMgrHome myBankMgrHome;
   private AccountHome myAccountHome;

   /** Creates new BankAccess */
   public BankAccess() {}

   public static void main (String args[]) {
      BankAccess ba = new BankAccess();
      try {
         ba.init();
         String myID = null;
         BankMgr myBankMgr = ba.myBankMgrHome.create();
         myID = myBankMgr.loginUser("Al", "password");

         System.out.println("User ID for user Al is: " + myID);
         CustomerData myCD = myBankMgr.getCustomerData(myID);
         System.out.println("myCD is: " + myCD);
         CustomerWithAcctsData myCWAD =
            myBankMgr.getCustomerWithAcctsData(myID);
         System.out.println("myCWAD is: " + myCWAD);
         myBankMgr.remove();

         AccountData myAccountData = null;
         Account myAccount =
            ba.myAccountHome.create(myID, "discover", 20000.00);
         myAccountData = myAccount.getAccountData();
         System.out.println("\nThe account data for account # " +
                            myAccountData.getAcctNumber() +
                            " is: " + myAccountData);
         myAccount.remove();
      } catch (Exception e) {
         System.out.println("Error in BankAccess.main(): " + e);
      }
   }

    public void init() { ... }

    private void initMyBankMgrHome() { ... }

    private void initMyAccountHome() { ... }
}

************************************************************
```

Answers for Discussion Topics:

- What other mechanisms could you use to generate a unique primary key when adding a row to a database table?

  Answer: Several mechanisms could be used including a random number generator or a native database utility.

- Discuss when using an entity EJB to generate unique primary keys might be inappropriate.

  Answer: Using an entity EJB to maintain unique primary keys only works when it controls primary key generation for all row additions across all applications.

- How would implementing a BMP `Sequencer` EJB's home and remote interfaces differ from the CMP implementation used in this lab?

  Answer: There is no difference between how the home and remote interfaces are defined when using BMP or CMP.

- How would implementing a BMP `Sequencer` EJB's bean class differ from the CMP implementation used in this lab?

  Answer: The bean developer would have to supply the data access code for all of the bean class methods that access the application data store.

- How does the container determine which of the `Account` EJB's two creator methods to invoke when a client application requests that a new row be added to the Account data store?

  Answer: The container invokes the creator method with the method signature that corresponds to the method called by the client.

- What method does an EJB client use to delete a row from the application data store?

  Answer: The EJB client invokes the bean's `remove` method.

Filename:	lab08.doc
Directory:	H:\FJ310_revC_0301\BOOK\EXERCISE\Exercise_Originals
Template:	C:\Program Files\Microsoft Office\Templates\Normal.dot
Title:	Case Study: The New User Interface
Subject:	
Author:	jdibble
Keywords:	
Comments:	
Creation Date:	04/24/01 10:17 AM
Change Number:	2
Last Saved On:	04/24/01 10:17 AM
Last Saved By:	Nancy Clarke
Total Editing Time:	1 Minute
Last Printed On:	04/24/01 12:38 PM

As of Last Complete Printing

Number of Pages:	18
Number of Words:	2,994 (approx.)
Number of Characters:	17,069 (approx.)