Exercise 4:

Creating a BMP Entity EJB™

Case Study: Creating a BMP Entity EJB

Estimated completion time: 1 hour 15 minutes.

In this exercise, you create a BMP entity EJB™ to access the banking application's customer database table and retrieve customer data. The `Customer` EJB provides read-only access to the customer table. The customer table is named `J2EECUSTOMER` and contains information about each banking customer. The following snippet of SQL code was used to create the `J2EECUSTOMER` table:

```
CREATE TABLE J2EECUSTOMER(
      CUSTOMERID   VARCHAR(6) CONSTRAINT PK_J2EECUSTOMER PRIMARY KEY,
      FIRSTNAME    VARCHAR(20),
      LASTNAME     VARCHAR(20),
      ADDRESS      VARCHAR(50),
      PHONE        VARCHAR(20));
```

The Customer EJB contains fields that map to corresponding columns in the entity data store. The `Customer` EJB also provides a mechanism to search for a specific customer based on the customer's ID code and contains a single business method that returns a customer data model object for a specified customer. All of the SQL code required to interface with the database is contained within the Customer EJB's bean class for simplicity. It is more common and preferable to encapsulate the data access code in a separate Data Access Object. As shown in Figure 1, the `BankMgr` session EJB serves as the EJB tier interface to the functionality provided by the `Customer` entity EJB. It is quite common to use session EJBs that encapsulate an application's business logic as the interface to EJB tier components.

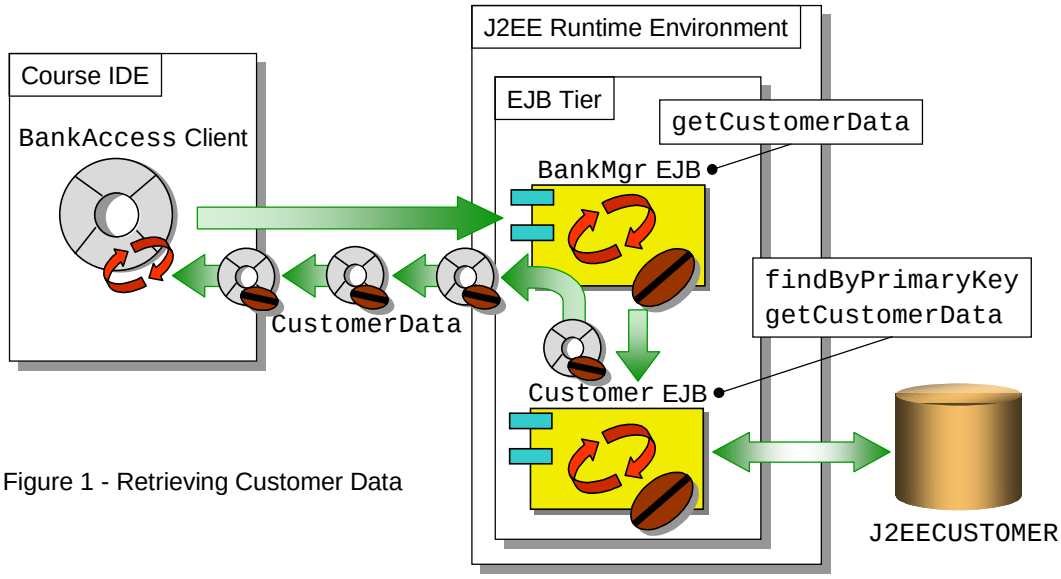

Figure 1 - Retrieving Customer Data

There are three sections to this lab exercise. In the first section, you code selected portions of the `Customer` BMP entity EJB. In Section 2, you modify the `BankAccess` test client and `BankMgr` session EJB to invoke the functionality of the `Customer` BMP entity EJB. Finally, in Section 3, you deploy and test the application components.

After completing this lab, you will be able to:

- Describe the components of a BMP entity EJB
- Explain how to create a BMP entity EJB using the course IDE
- Describe the mechanism used by a BMP entity EJB to connect with a data store

- List the steps necessary to deploy a BMP entity EJB into the J2EE runtime environment
- Describe how to access a BMP entity EJB from a session EJB client

Discussion Topics:

- Discuss the role of an entity EJB's persistent fields.
- What general classification of methods implemented in an entity EJB's bean class require corresponding method stubs on the remote interface?
- What methods must be implemented on an entity EJB's bean class for which method stubs do not appear on the EJB's home or remote interfaces?

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.sql

```
   public interface Statement
      public ResultSet executeQuery(String sql) throws SQLException
      public void close() throws SQLException
      public int executeUpdate(String sql) throws SQLException
   public interface ResultSet
      public boolean next() throws SQLException
      public void close() throws SQLException
      double getDouble(int columnIndex) or double getDouble(String
                                    columnName)
      int getInt(int columnIndex) or int getInt(String columnName)
      String getString(int columnIndex) or String getString(String
                                    columnName)
   public interface Connection
      public Statement createStatement() throws SQLException
      public PreparedStatement prepareStatement(String sql) throws
                                    SQLException
      public void close() throws SQLException
```

javax.ejb

```
   public class CreateException extends java.lang.Exception
   public class EJBException extends java.lang.RuntimeException
   public class FinderException extends java.lang.Exception
   public class RemoveException extends java.lang.Exception
   public interface EntityContext extends EJBContext
      java.lang.Object getPrimaryKey()
```

javax.naming

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

javax.rmi

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

javax.sql

```
public interface DataSource
    java.sql.Connection getConnection() throws java.sql.SQLException
```

Setup Instructions

The files containing the skeleton code for the `Customer` entity EJB created in this exercise are located in the course resources directory. The `BankAccess` class serves as the test client for this lab. Use the "Guidelines for Creating a BMP 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 `Customer` BMP Entity EJB's Home and Remote Interface

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

1. Create a home interface for the `Customer` EJB in the `J2EEApp` package. The `Customer` EJB does not require the ability to add a row to the customer data store and, therefore, does not define a creator method. The `Customer` EJB provides all of its searching capabilities based on the mandatory `findByPrimaryKey` method.
   - Name: `CustomerHome`
   - Creator/finder methods:
     - `findByPrimaryKey(String customerID)`– Locates a customer based on an identifying code
2. Create a remote interface for the `Customer` EJB in the `J2EEApp` package.
   - Name: `Customer`
   - Business methods:
     - `CustomerData getCustomerData` – Creates a model of the current customer

Verify Your Work

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

Import/Update the `Customer` BMP Entity EJB's Bean Class

A partially completed bean class for the `Customer` EJB is furnished with the lab resources for this course as described in the setup section of this lab exercise. Follow the steps listed below to finish implementing this class.

4. Locate the file `CustomerEJB.java` in the course resource directory, and copy it to the `J2EEApp` package in the IDE workspace. Review the code. Notice that several of the class methods are incomplete. The code requiring modification has been delimited with the comments "begin: insert new code" and "end: insert new code". You can locate these sections of code using the editor's search tools.
5. Code each of the methods in the `Customer` EJB's bean class listed below using the comments provided in each method as a guideline for completing the required code. Use the completed methods found on the `Customer` EJB's bean class as an example of how to implement some of the missing functionality.
   - `loadRow`
   - `ejbStore`
   - `selectByPrimaryKey`

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 does the operation of an entity EJB's creator method differ from that of a session EJB?
- What elements of an entity EJB correspond to the columns in the associated data store?
- What, if any, methods must be implemented by a BMP entity EJB developer that are accessed using the home interface?
- When must an entity EJB developer implement a creator method?
- What is the function of an entity bean's primary key class?
- When must a bean developer create a custom primary key class for an entity bean?
- What can be done to improve the efficiency of the `ejbStore` method?

Section 2

Modify the `BankMgr` Session EJB

The `BankMgr` session EJB provides the mechanism for accessing the functionality provided by the `Customer` EJB. In this section, you add a method to the `BankMgr` session bean to retrieve a `CustomerData` model object for a specified customer.

1. Modify the `BankMgr` session EJB's remote interface as follows:
   - Business methods:
     - `CustomerData getCustomerData(String customerID)`
2. Perform the following modifications to the `BankMgr` EJB's bean class:
   - Add a field to the bean class to store the `Customer` EJB's home interface reference:
     - Name: `myCustomerHome`
     - Visibility: `private`
     - Type: `CustomerHome`
   - Update the `ejbCreate` method to locate the `Customer` EJB's home interface in the J2EE namespace using JNDI and store the reference in `myCustomerHome`. The `Customer` EJB is registered as "customerServer".
   - Add a method to the `BankMgr` bean class to implement the `getCustomerData` method previously added to the EJB's remote interface:
     - Name: `getCustomerData`
     - Visibility: `public`
     - Returns: `CustomerData`
     - Parameters:
       - `String customerID`
     - Functionality: Code the method to locate a row in the customer table associated with the parameter `customerID` using the `Customer` EJB's `findByPrimaryKey` method, and return a `CustomerData` model object for the customer by invoking the `Customer` EJB's `getCustomerData` method for the specified customer.

Modify the BankAccess Test Client

The BankAccess class serves as the test client for this exercise. The BankAccess client does not interact with the Customer EJB directly. The BankAccess class uses the methods available on the BankMgr session EJB to access the functionality provided by the Customer EJB. For this lab, the BankAccess client retrieves a CustomerData object for a specified customer using the BankMgr EJB's getCustomerData method. To access the getCustomerData method on BankMgrEJB, you modify the BankAccess class's main method to create a BankMgr instance and invoke the EJB's getCustomerData method.

3. Modify the BankAccess's main method to call the BankMgr EJB's getCustomerData method and retrieve customer information for one or more bank customers as follows:

   - Create a new BankMgrEJB instance
   - Call the BankMgrEJB's getCustomerData method to retrieve a CustomerData object for at least one specified customer and print the results
   - Call BankMgrEJB's remove method

Compile

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

Discussion Topics:

- What benefits are obtained by accessing the functionality of the Customer entity EJB using the BankMgr session EJB as opposed to invoking the Customer EJB's methods directly from the BankAccess client?

Section 3

Deploy and Test

In this section, you deploy and test the `Customer` entity EJB. Consult the "Assembly and Deployment Guidelines" for information on deploying EJBs 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 `Customer` entity EJB into the J2EE runtime environment. Use "customerServer" as the JNDI name for the `Customer` EJB and "jdbc/BankMgrDB" as the JNDI name for the database factory resource. 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.`
     - Add the `Customer` 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 `Customer.class`, `CustomerData.class`, `CustomerEJB.class`, and `CustomerHome.class` from the `J2EEApp` package.
       - **New Enterprise Bean Wizard - General**
         - Enterprise Bean Class: `J2EEApp.CustomerEJB`
         - Home Interface: `J2EEApp.CustomerHome`
         - Remote Interface: `J2EEApp.Customer`
         - Display Name: `CustomerBean`
         - Bean Type: Entity
       - **New Enterprise Bean Wizard - Entity Settings**
         - Bean-Managed Persistence: Checked
         - Primary Key Class: `java.lang.String`
       - **New Enterprise Bean Wizard - Resource References**
         - Resource Factories Referenced in Code:
           - `jdbc/BankMgrDB`; `javax.sql.DataSource`; Container

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

- Deploy the application:
  - Select the option to Return Client Jar, and store the client `.jar` file in the *<J2EEAppHome>* directory.
  - Use “bankmgrServer” and “customerServer” as the JNDI names for the `BankMgr` session and `Customer` entity EJBs, respectively.
  - Map the resource factory reference (`jdbc/BankMgrDB`) for the `BankMgr` 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

```
// CustomerHome.java

package J2EEApp;

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

public interface CustomerHome extends EJBHome {

   public Customer findByPrimaryKey (String customerID)
      throws RemoteException, FinderException;
}

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

// Customer.java

package J2EEApp;

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

public interface Customer extends EJBObject {

   public CustomerData getCustomerData() throws RemoteException;
}

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

// CustomerEJB.java

package J2EEApp;

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

public class CustomerEJB implements EntityBean {

   private String customerID;
   private String firstName;
   private String lastName;
   private String address;
   private String phone;

   private EntityContext context;

   public CustomerEJB() {}

   public void ejbRemove() throws RemoveException {}

   public void setEntityContext(EntityContext ec) {
      System.out.println("Entering CustomerEJB.setEntityContext()");
      this.context = ec;
      System.out.println("Leaving CustomerEJB.setEntityContext()");
```

```
}

public void unsetEntityContext() {
   System.out.println("Entering CustomerEJB.unsetEntityContext()");
   System.out.println("Leaving CustomerEJB.unsetEntityContext()");
}

public void ejbActivate() {
   System.out.println("Entering CustomerEJB.ejbActivate()");
   customerID = (String)context.getPrimaryKey();
   System.out.println("Leaving CustomerEJB.ejbActivate()");
}

public void ejbPassivate() {
   System.out.println("Entering CustomerEJB.ejbPassivate()");
   customerID = null;
   System.out.println("Leaving CustomerEJB.ejbPassivate()");
}

public void ejbLoad() {
   System.out.println("Entering CustomerEJB.ejbLoad()");
   String customerID = (String)context.getPrimaryKey();
   try {
      loadRow();
   } catch (Exception ex) {
      throw new EJBException("ejbLoad: " + ex.getMessage());
   }
   System.out.println("Leaving CustomerEJB.ejbLoad()");
}

private void loadRow() throws SQLException {
   System.out.println("Entering CustomerEJB.loadRow()");
   Connection dbConnection = initDBConnection();
   Statement stmt = dbConnection.createStatement();
   String queryStr = "SELECT firstname, lastname, address, phone "
      + " FROM J2EECUSTOMER WHERE customerid = "
      + "'" + this.customerID + "'";
   System.out.println("queryString is: "+ queryStr);
   ResultSet result = stmt.executeQuery(queryStr);
   if (result.next()) {
      int i = 1;
      firstName = (result.getString(i++)).trim();
      lastName = (result.getString(i++)).trim();
      address = (result.getString(i++)).trim();
      phone = (result.getString(i++)).trim();
      stmt.close();
      dbConnection.close();
   }
   else {
      stmt.close();
      dbConnection.close();
      throw new NoSuchEntityException("Row for id " + customerID +
                  " not found in database.");
   }
   System.out.println("Leaving CustomerEJB.loadRow()");
}

public void ejbStore() {
```

```
      System.out.println("Entering CustomerEJB.ejbStore()");
      customerID = (String)context.getPrimaryKey();
      try {
         storeRow();
      } catch (Exception ex) {
         throw new EJBException("ejbLoad: " + ex.getMessage());
      }
      System.out.println("Leaving CustomerEJB.ejbStore()");
   }

   private void storeRow() throws SQLException {
      System.out.println("Entering CustomerEJB.storeRow()");
      Connection dbConnection = initDBConnection();
      Statement stmt = dbConnection.createStatement();
      String queryStr = "UPDATE J2EECUSTOMER SET"
         + " firstname = '" +  firstName.trim()
         + "', lastname = '" + lastName.trim()
         + "', address = '" + address.trim()
         + "', phone = '" + phone.trim()
         + "' WHERE customerid = " + "'" + customerID + "'";
      System.out.println("queryString is: "+ queryStr);
      int resultCount = stmt.executeUpdate(queryStr);
      stmt.close();
      dbConnection.close();
      if (resultCount == 0) {
         throw new EJBException("Storing row for customerID " +
                                customerID + " failed.");
      }
      System.out.println("Leaving CustomerEJB.storeRow()");
   }

   public String ejbFindByPrimaryKey (String key)
         throws FinderException {
      System.out.println("Entering CustomerEJB.ejbFindByPrimaryKey()");
      customerID = key;
      try {
      if (!selectByPrimaryKey(customerID))
         throw new ObjectNotFoundException("Row for id " + key +
                                           " not found.");
      } catch (Exception ex) {
         throw new EJBException("ejbFindByPrimaryKey: " +
                                ex.getMessage());
      }
      System.out.println("Leaving CustomerEJB.ejbFindByPrimaryKey()");
      return customerID;
   }

   private boolean selectByPrimaryKey(String key) throws SQLException {
      System.out.println("Entering CustomerEJB.selectByPrimaryKey()");
      Connection dbConnection = initDBConnection();
      Statement stmt = (this.dbConnection).createStatement();
      String queryStr =
         "SELECT customerid FROM J2EECUSTOMER WHERE customerid = "
            + "'" + key + "'";
      System.out.println("queryString is: "+ queryStr);
      ResultSet rs = stmt.executeQuery(queryStr);
      boolean result = rs.next();
      stmt.close();
```

```
      dbConnection.close();
      System.out.println("Leaving CustomerEJB.selectByPrimaryKey()");
      return result;
   }

   // business methods
   public CustomerData getCustomerData() {
      return (new CustomerData(customerID,firstName,lastName,phone));
  }

   // utility method
   private Connection initDBConnection() throws SQLException {
      System.out.println("Entering CustomerEJB.initDBConnection()");
      Connection c = null;
      try {
         InitialContext ic = new InitialContext();
         DataSource ds =
            (DataSource)ic.lookup("java:comp/env/jdbc/BankMgrDB");
         c = ds.getConnection();
      } catch (NamingException ne) { throw new EJBException(ne); }
      catch (SQLException se) { throw new EJBException(se); }
      System.out.println("Leaving CustomerEJB.initDBConnection()");
      return c;
   }
}

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

Solution Code: Section 2

// BankMgr.java

```
package J2EEApp;

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

public interface BankMgr extends EJBObject {

   public String loginUser(String pUsername, String pPassword)
      throws RemoteException;

   public CustomerData getCustomerData(String customerID)
      throws RemoteException;
}

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

// BankMgrEJB.java

```
package J2EEApp;

import java.rmi.RemoteException;

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

public class BankMgrEJB implements SessionBean {

   private javax.sql.DataSource jdbcFactory;
   private CustomerHome myCustomerHome;

   public BankMgrEJB() {}

   public void ejbCreate() {
      System.out.println("Entering BankMgrEJB.ejbCreate()");
      Context c = null;
      Object result = null;
      if (this.jdbcFactory == null) {
         System.out.println("jdbcFactory is null in ejbCreate()");
         try {
            c = new InitialContext();
            this.jdbcFactory = (javax.sql.DataSource)
              c.lookup("java:comp/env/jdbc/BankMgrDB");
         } catch (Exception e){ System.out.println("Error: " + e); }
      }
      if (this.myCustomerHome == null) {
         try {
            c = new InitialContext();
            result = c.lookup("customerServer");
            myCustomerHome =
            (CustomerHome)javax.rmi.PortableRemoteObject.narrow
               (result,CustomerHome.class);
         }
```

```
            catch (Exception e) { System.out.println("Error: " + e); }
         }
         System.out.println("Leaving BankMgrEJB.ejbCreate()");
      }

      public String loginUser(String pUsername, String pPassword) {
         System.out.println("Entering BankMgrEJB.loginUser()");
         String theID = null;
         try {
            Connection conn = this.jdbcFactory.getConnection();
            Statement  stmt = conn.createStatement();
            String sqlString =
               "select UserID from J2EEAPPUSER where Username = '" +
               pUsername + "' and Password = '" + pPassword + "'";
            System.out.println(sqlString);
            ResultSet rs = stmt.executeQuery(sqlString);
            if ( rs.next() ) {
               theID = rs.getString("UserID");
               System.out.println("userID is: " + theID);
            }
            rs.close();
            stmt.close();
            conn.close();
         } catch ( Exception e ) { System.out.println("Error: " + e); }
         System.out.println("Leaving BankMgrEJB.loginUser()");
         return theID;
      }

      public CustomerData getCustomerData(String customerID) {
         System.out.println("Entering BankMgrEJB.getCustomerData()");
         CustomerData myCD = null;
         try {
            Customer myCustomer =
               myCustomerHome.findByPrimaryKey(customerID);
            if (myCustomer != null) {
               myCD = myCustomer.getCustomerData();
               System.out.println("Customer data is: "+ myCD);
            }
         }
         catch (Exception e) {
            System.out.println
               ("Error in BankMgrEJB.getCustomerData(): " + e);
         }
         System.out.println("Leaving BankMgrEJB.getCustomerData()");
         return myCD;
      }

      public void setSessionContext(SessionContext sc) {}
      public void ejbRemove() {}
      public void ejbActivate() {}
      public void ejbPassivate() {}
   }

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

   // BankAccess.java

   package J2EEApp;
```

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;

public class BankAccess extends Object {

   private BankMgrHome myBankMgrHome;

   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);
         myBankMgr.remove();
      }
      catch (Exception e) {
         System.out.println("Error in BankAccess.main(): " + e);
      }
   }

   public void init() {
      System.out.println("Entering BankAccess.init()");
      try {
         if (myBankMgrHome == null) initMyBankMgrHome();
      } catch (Exception e) {
         System.out.println("Error in BankAccess.init(): " + e);
      }
      System.out.println("Leaving BankAccess.init()");
   }

   private void initMyBankMgrHome() {
      System.out.println("Entering BankAccess.initMyBankMgrHome()");
      try {
         Context c = new InitialContext();
         Object result = c.lookup("bankmgrServer");
         myBankMgrHome =
            (BankMgrHome)javax.rmi.PortableRemoteObject.
               narrow(result,BankMgrHome.class);
      }
      catch (Exception e) { System.out.println(e); }
      System.out.println("Leaving BankAccess.initMyBankMgrHome()");
   }
}
```

Answers for Discussion Topics:

- Discuss the role of an entity EJB's persistent fields.

  Answer: An entity EJB's persistent fields map to the associated column elements in the persistent data store.

- What general classification of methods implemented in an entity EJB's bean class require corresponding method stubs on the remote interface?

  Answer: Application business methods and any accessor methods required by the EJB clients.

- What methods must be implemented on an entity EJB's bean class for which method stubs do not appear on the EJB's home or remote interfaces?

  Answer: Container-invoked methods `ejbLoad`, `ejbStore`, `setEntityContext`, `ejbRemove`, `ejbPassivate`, and `ejbActivate`.

- How does the operation of an entity EJB's creator method differ from that of a session EJB?

  Answer: The creator method on an entity EJB is used to add a new row to the data store. The parameters of an entity EJB's creator method provide some or all of the column values for the new row. A session EJB's creator method is used to create a session bean instance.

- What elements of an entity EJB correspond to the columns in the associated data store?

  Answer: The EJB's persistent fields.

- What, if any, methods must be implemented by a BMP entity EJB developer that are accessed using the home interface?

  Answer: The home interface contains method stubs for an EJB's creator and finder methods. The bean developer must implement any creator or finder methods defined on the home interface. A BMP entity EJB must define and implement a `findByPrimaryKey` method.

- When must an entity EJB developer implement a creator method?

  Answer: An entity EJB requires a creator method to add a new row to the data store. If adding a row to the data store is part of the application specifications, then a bean developer must implement one or more creator methods for the EJB.

- What is the function of an entity bean's primary key class?

  Answer: The container uses the primary key class to assign a unique identity to EJB instances. The primary key class must implement `hashCode` and `equals` methods. In most cases, an EJB's primary key is the persistent field associated with the database column serving as the primary key for the table.

- When must a bean developer create a custom primary key class for an entity bean?

  Answer: A standard Java class can usually suffice as a primary key class as long as it implements the `hashCode` and `equals` methods. For example, a persistent field defined as a `String` or `Integer` object can serve as an EJB's primary key class. A bean developer must implement a primary key class for complex primary keys that use multiple fields or fields defined by classes that do not implement a `hashCode` or an `equals` method.

- What can be done to improve the efficiency of the `ejbStore` method?

  Answer: The bean class can be modified to include a mechanism to prevent `ejbStore` from writing to the database if the bean's persistent fields have not been changed.

- What benefits are obtained by accessing the functionality of the `Customer` entity EJB using the `BankMgr` session EJB as opposed to invoking the `Customer` EJB's methods directly from the `BankAccess` client?

  Answer: Session beans are normally used to encapsulate the business logic associated with an application while entity EJBs represent the application data model. In many cases, a business method must access a suite of entity EJBs to perform a transaction. For example, when adding a new customer to a banking application, a new row must be added to the appuser, customer, and account tables. Delegating the application business logic as well as responsibility for coordinating entity bean access and transaction management to a session bean is generally more desirable than burdening the application client with these tasks.

Filename:	lab04.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 10:08 AM
Change Number:	2
Last Saved On:	04/24/01 10:08 AM
Last Saved By:	Nancy Clarke
Total Editing Time:	0 Minutes
Last Printed On:	04/24/01 12:37 PM

As of Last Complete Printing

Number of Pages:	20
Number of Words:	3,909 (approx.)
Number of Characters:	22,284 (approx.)