Exercise 5:

Using Data Access Objects With BMP Entity EJB™

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Case Study: Using Data Access Objects With BMP Entity EJBs

Estimated completion time: 1 hour 30 minutes.

In this exercise, you create a BMP entity EJB™ to access the banking application's `J2EEACCOUNT` database table. The `J2EEACCOUNT` database table contains account information. The `J2EEACCOUNT` table was created using the following SQL script:

```
CREATE TABLE J2EEACCOUNT(
      ACCTNUMBER  VARCHAR(6) CONSTRAINT PK_J2EEACCOUNT PRIMARY KEY,
      CUSTOMERID  VARCHAR(6),
      DESCRIPTION VARCHAR(20),
      BALANCE     DECIMAL(12,2),
         CONSTRAINT FK_J2EEACCOUNT_1 FOREIGN KEY (CUSTOMERID)
            REFERENCES J2EECUSTOMER (CUSTOMERID));
```

The `Account` entity EJB provides a mechanism to search for a specific account based on an account ID code and contains a single business method that returns an account data model object for a specified account. The `Account` EJB also contains methods to add new accounts to and delete accounts from the `J2EEACCOUNT` table. In this application, the `BankAccess` client bean accesses the functionality provided by the `Account` EJB directly. Accessing entity EJB's directly from a simple Java client is generally not recommended practice. It is done here to reduce the amount of work required to complete the lab exercise.

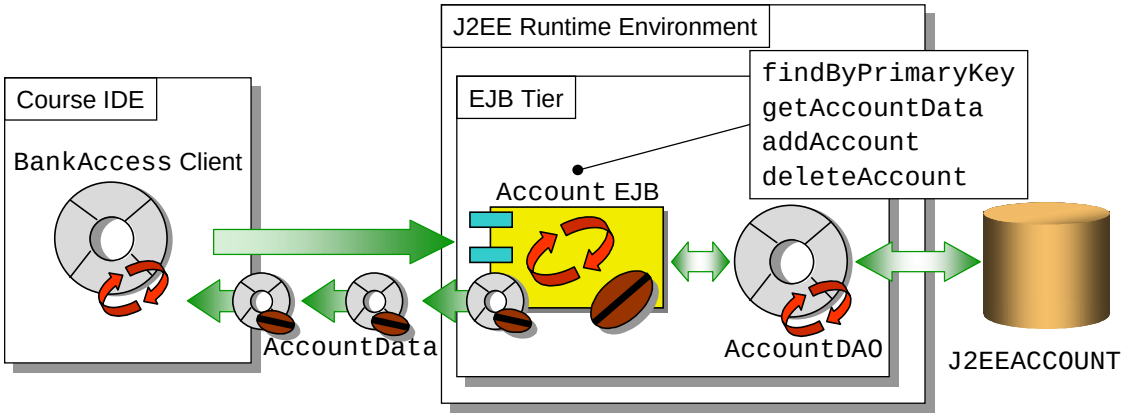

Figure 1 - The `Account` Entity EJB

The `Account` BMP entity EJB uses a Data Access Object (DAO) to encapsulate the SQL code required to access the `J2EEACCOUNT` table. A DAO contains fields and functionally equivalent methods to those supplied on the associated EJB. For example, the data access code to implement an EJB's `ejbCreate` and `ejbRemove` methods must be supplied by corresponding methods on the Data Access Object. A DAO must also implement database access code for an entity EJB's business and finder methods, such as `getAccountData` and `ejbFindByPrimaryKey`. The Data Access Object used in this lab exercise contains a suite of public methods corresponding to the methods found on the `Account` EJB. Each of these public methods invoke one or more private methods that implement the actual data access code. For example, the `Account` EJB's `ejbFindByPrimaryKey` method calls the DAO's `findByPrimaryKey` method that in turn invokes the private method `accountExists` that executes the SQL code required to locate the desired account in the data store.

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

There are two sections to this lab exercise. In the first section, you code selected portions of the `Account` BMP entity EJB using the functionality provided by the `AccountDAO`. You also implement one of the Data Access Object's methods. In Section 2, you first modify the `BankAccess` test client to invoke the functionality of the `Account` BMP entity EJB and then deploy and test the application.

After completing this lab, you will be able to:

- Describe the function of a Data Access Object
- Implement a BMP entity EJB's bean class using a Data Access Object for data store access
- Create the Java code required to connect to a database and perform simple database access functions
- Lists the steps required to deploy a BMP entity EJB and associated Data Access Object into the J2EE runtime environment

Discussion Topics:

- What methods contained in an EJB's bean class can benefit from using the functionality found on a Data Access Object?
- What SQL commands are used to add a new row to a database table, remove a row from a database table, and modify one or more data elements in a row of a database table?

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

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
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Setup Instructions

The files containing the skeleton code for some of the classes used 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", the "Guidelines for Using BMP Entity EJBs with Data Access Objects", 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.

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Section 1

Define the `Account` 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 `Account` EJB in the `J2EEApp` package.
   - Name: `AccountHome`
   - Creator/finder methods:
     - `create(String acctNumber, String customerID, String description, double balance)`
     - `findByPrimaryKey(String acctNumber` – Locates an account based on an identifying code
2. Create a remote interface for the `Account` EJB in the `J2EEApp` package.
   - Name: `Account`
   - Business methods:
     - `AccountData getAccountData` – Creates a data model object of the current account

Verify Your Work

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

Discussion Topics:

- Describe how using a Data Access Object to encapsulate the beans data access code affects the home and remote interface definitions.

Import/Update the `Account` BMP Entity EJB's Bean Class and DAO

A partially completed bean class for the `Account` EJB and associated Data Access Object 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 files `AccountEJB.java` and `AccountDAO.java` in the course resource directory, and copy them to the `J2EEApp` package in the IDE workspace. Review the code. 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 `Account` 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 `Account` EJB's bean class as an example of how to implement some of the missing functionality.
   - `ejbCreate`
   - `ejbRemove`
   - `ejbLoad`
   - `ejbStore`

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

6. Code each of the methods in the `AccountDAO` 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 `AccountDAO` class as an example of how to implement some of the missing functionality.
   - `selectAccount`
   - `deleteAccount` – Code is similar to `updateAccount`

Compile

7. 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:

- Discuss the design goals behind not embedding a DAO's database access logic in the public methods invoked by the EJB.

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Section 2

Modify the `BankAccess` Client

The `BankAccess` class serves as the test client in this exercise. To test the `Account` EJB's functionality, you modify the `BankAccess` class to exercise the methods on the `Account` EJB.

1. Add a field to the `BankAccess` class used to store the `Account` EJB's home interface reference.
   - Name: `myAccountHome`
   - Visibility: `private`
   - Type: `AccountHome`
2. Add a method to the `BankAccess` class.
   - Name: `initMyAccountHome`
   - Return Type: `void`
   - Visibility: `private`
   - Parameters: None
     - Functionality: Code the method to lookup the `Account` EJB's home interface using the JNDI name "accountServer", and store the home interface reference to the field `myAccountHome`. This method is similar to `initMyBankMgrHome` added to the `BankAccess` class in an earlier exercise.
3. Update the `BankAccess`'s `init` method to invoke `initMyAccountHome`.
4. Update the `BankAccess`'s `main` method to:
   - Create a new account using the following data elements:
     - account number: 1000
     - customer ID: use the ID for customer Al; `username=='Al'`, `password=='password'`
     - account type: `discover`
     - account balance: $10000.00
   - Retrieve an `AccountData` model object for the newly created account, and output its contents to the IDE's run console.
   - Delete the newly added account.

Compile

5. Compile the `BankAccess` class. Correct any errors before continuing with this exercise.

Discussion Topics:

- How does implementing an entity bean's data access code in a DAO impact an EJB client?
- How does implementing an entity bean's data access code in a DAO impact the steps taken when the bean is deployed?

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Section 3

Deploy and Test

In this section, you deploy and test the `Account` entity EJB. 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 `Account` entity EJB into the J2EE runtime environment. Use "bankmgrServer", "customerServer", and "accountServer" as the JNDI names for the `BankMgr` session bean and the `Customer` and `Account` entity EJBs, 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`.
     - Add the `Account` 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 `Account.class`, `AccountData.class`, `AccountDAO.class`, `AccountEJB.class`, and `AccountHome.class` from the `J2EEApp` package.
       - **New Enterprise Bean Wizard - General**
         - Enterprise Bean Class: `J2EEApp.AccountEJB`
         - Home Interface: `J2EEApp.AccountHome`
         - Remote Interface: `J2EEApp.Account`
         - Display Name: `AccountBean`
         - 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

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

- **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”, “customerServer”, and “accountServer” as the JNDI names for the `BankMgr` session bean and the `Customer` and `Account` entity EJBs, respectively.
  - Map the resource factory reference (`jdbc/BankMgrDB`) for the `BankMgr`, `Customer`, and `Account` beans 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.

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Solution Code: Section 1

```
// 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 findByPrimaryKey (String key)
        throws RemoteException, FinderException;
}

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

// Account.java

package J2EEApp;

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

public interface Account extends EJBObject {

   public AccountData getAccountData() throws RemoteException;
}

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

// AccountEJB.java

package J2EEApp;

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


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 acctNumber, String customerID,
                            String description, double balance)
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
    throws DuplicateKeyException,CreateException {

    System.out.println("Entering AccountEJB.ejbCreate()");
    this.acctNumber = acctNumber;
    this.customerID = customerID;
    this.description = description;
    this.balance = balance;
    AccountDAO ADAO = new AccountDAO();
    ADAO.setAcctNumber(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()");
    return (acctNumber);
}

public void ejbRemove() throws RemoveException {
    System.out.println("Entering AccountEJB.ejbRemove()");
    AccountDAO ADAO = new AccountDAO();
    ADAO.setAcctNumber((String)context.getPrimaryKey());
    try{
       Connection dbConnection = getDBConnection();
       ADAO.remove(dbConnection);
       dbConnection.close();
    } catch (java.sql.SQLException se) {
       throw new RemoveException ("SQL Exception in remove");
    }
    System.out.println("Leaving AccountEJB.ejbRemove()");
}

public void ejbLoad() {
    System.out.println("Entering AccountEJB.ejbLoad()");
    AccountDAO ADAO = new AccountDAO();
    ADAO.setAcctNumber((String)context.getPrimaryKey());
    try{
       Connection dbConnection = getDBConnection();
       ADAO.load(dbConnection);
       dbConnection.close();
    } catch (java.sql.SQLException se) {
       throw new EJBException (se);
    }
    acctNumber = ADAO.getAcctNumber();
    customerID = ADAO.getCustomerID();
    description = ADAO.getDescription();
    balance = ADAO.getBalance();
    System.out.println("Leaving AccountEJB.ejbLoad()");
}

public void ejbStore() {
    System.out.println("Entering AccountEJB.ejbStore()");
    AccountDAO ADAO = new AccountDAO();
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
      ADAO.setAcctNumber((String)context.getPrimaryKey());
      ADAO.setCustomerID(customerID);
      ADAO.setDescription(description);
      ADAO.setBalance(balance);
      try{
         Connection dbConnection = getDBConnection();
         ADAO.store(dbConnection);
         dbConnection.close();
      } catch (java.sql.SQLException se) {
         throw new EJBException (se);
      }
      System.out.println("Leaving AccountEJB.ejbStore()");
   }


   public String ejbFindByPrimaryKey (String key)
      throws FinderException {
      System.out.println("Entering AccountEJB.ejbFindByPrimaryKey()");
      AccountDAO ADAO = new AccountDAO();
      ADAO.setAcctNumber(key);
      String theKey = null;
      try{
         Connection dbConnection = getDBConnection();
         theKey = ADAO.findByPrimaryKey(dbConnection);
         dbConnection.close();
      } catch (java.sql.SQLException se) {
         throw new FinderException
                     ("SQL Exception in find by primary key");
      }
      System.out.println("Leaving AccountEJB.ejbFindByPrimaryKey()");
      return theKey;
   }


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

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

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

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

   public void ejbPostCreate(String acctNumber, String customerID,
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
                                String description, double balance)
      throws DuplicateKeyException,CreateException {
   }

   // business methods
   public AccountData getAccountData() {
      return (new AccountData(acctNumber, balance, description));
   }

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

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

// AccountDAO.java

```
package J2EEApp;


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

public class AccountDAO extends Object {

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

   private Connection dbConnection = null;

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

   // get and set methods for the instance variables
   public String getAcctNumber(){ return acctNumber; }
   public void setAcctNumber(String acctNumber){
      this.acctNumber = acctNumber;
   }

   public String getCustomerID(){ return customerID; }
   public void setCustomerID(String customerID){
    this.customerID = customerID;
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
}

public String getDescription(){ return description; }
public void setDescription(String description){
   this.description = description;
}

public double getBalance(){ return balance; }
public void setBalance(double balance){
   this.balance = balance;
}

// public methods
public void load ( Connection con ) throws SQLException {
   dbConnection = con;
   selectAccount();
}

public void store( Connection con ) throws SQLException {
   dbConnection = con;
   updateAccount();
}

public void remove( Connection con ) throws SQLException {
   dbConnection = con;
   deleteAccount();
}

public void create(Connection con)
   throws SQLException, DuplicateKeyException {
   dbConnection = con;
   insertAccount();
}

public String findByPrimaryKey(Connection con) throws SQLException {
   dbConnection = con;
   if (accountExists(acctNumber)) return (acctNumber);
   throw new SQLException ("primary key not found:" + acctNumber);
}

// private methods
private boolean accountExists (String pAccountID)
   throws SQLException {
   Statement stmt = dbConnection.createStatement();
   String queryStr =
      "SELECT acctnumber FROM J2EEACCOUNT WHERE acctnumber = "
      + "'" + acctNumber + "'";
   System.out.println("queryString is: "+ queryStr);
   ResultSet result = stmt.executeQuery(queryStr);
   if (result.next()) {
      acctNumber = result.getString(1);
      stmt.close();
      return true;
   }
   else {
      stmt.close();
      return false;
   }
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
}

private void selectAccount () throws SQLException {
   System.out.println("Entering AccountDAO.selectAccount()");
   Statement stmt = dbConnection.createStatement();
   String queryStr =
      "SELECT acctnumber, customerid, description, balance "
      + " FROM J2EEACCOUNT WHERE acctnumber = "
      + "'" + acctNumber + "'";
   System.out.println("queryString is: "+ queryStr);
   ResultSet result = stmt.executeQuery(queryStr);
   if ( !result.next() )
      throw new SQLException
                  ("No record for primary key " + acctNumber);
   int i = 1;
   acctNumber = (result.getString(i++)).trim();
   customerID = (result.getString(i++)).trim();
   description = (result.getString(i++)).trim();
   balance = result.getDouble(i++);
   stmt.close();
   System.out.println("Leaving AccountDAO.selectAccount()");
}

private void insertAccount()
   throws SQLException,DuplicateKeyException {
   System.out.println("Entering AccountDAO.insertAccount()");
   if (accountExists(acctNumber))
      throw new DuplicateKeyException
                  ("Account Exists:   " + acctNumber );
   Statement stmt = dbConnection.createStatement();
   String queryStr =
      "INSERT INTO J2EEACCOUNT"
      + "(acctnumber,customerid,description,balance)" + "VALUES ("
      + "'" + acctNumber.trim() + "',"
      + "'" + customerID.trim() + "',"
      + "'" + description.trim() + "'," + balance + ")";
   System.out.println("queryString is: "+ queryStr);
   int resultCount = stmt.executeUpdate(queryStr);
   if ( resultCount != 1 ) {
      System.out.println("Error in AccountDAO.insertAccount()");
      throw new SQLException
                  ("ERROR in account INSERT !! resultCount = " +
                     resultCount);
   }
   stmt.close();
   System.out.println("Leaving AccountDAO.insertAccount()");
}

private void deleteAccount() throws SQLException {
   System.out.println("Entering AccountDAO.deleteAccount()");
   Statement stmt = dbConnection.createStatement();
   String queryStr =
      "DELETE FROM J2EEACCOUNT WHERE acctnumber = "
      + "'" + acctNumber + "'";
   System.out.println("queryString is: "+ queryStr);
   int resultCount = stmt.executeUpdate(queryStr);
   if ( resultCount != 1 )
      throw new SQLException
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
                    ("ERROR deleting account!! resultCount = "
                       + resultCount);
      stmt.close();
      System.out.println("Leaving AccountDAO.deleteAccount()");
   }

   private void updateAccount() throws SQLException {
      Statement stmt = dbConnection.createStatement();
      String queryStr =
         "UPDATE J2EEACCOUNT"
         + " SET customerid = '" + customerID.trim()
         + "', description = '" + description.trim()
         + "', balance = " + balance
         + " WHERE acctnumber = " + "'" + acctNumber + "'";
      System.out.println("queryString is: "+ queryStr);
      int resultCount = stmt.executeUpdate(queryStr);
      if ( resultCount != 1 )
         throw new SQLException
                    ("ERROR updating account!! resultCount = "
                       + resultCount);
      stmt.close();
   }
}
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Solution Code: Section 2

// BankAccess.java

```
package J2EEApp;

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

public class BankAccess extends Object {

   private AccountHome myAccountHome;
   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();

         AccountData myAccountData = null;
         String nextValue = "1000";
         String userID = "1";
         String acctType = "discover";
         double theBalance = 10000.00;
         Account myAccount =
            ba.myAccountHome.create
               (nextValue,userID,acctType,theBalance);
         if (myAccount != null) {
            myAccountData = myAccount.getAccountData();
            System.out.println("Account data for account # " +
                               myAccountData.getAcctNumber() +
                               " is: " + myAccountData);
            myAccount.remove();

         }
         else System.out.println("Could not create new Account");
      }
      catch (Exception e) {
         System.out.println("Error in BankAccess.main(): " + e);
      }
   }

   public void init() {
      System.out.println("Entering BankAccess.init()");
      try {
         if (myAccountHome == null) initMyAccountHome();
         if (myBankMgrHome == null) initMyBankMgrHome();
      }
      catch (Exception e) {
```


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

```
            System.out.println("Error in BankAccess.init(): " + e);
         }
         System.out.println("Leaving BankAccess.init()");
      }

      private void initMyBankMgrHome() { ... }

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


Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Answers for Discussion Topics:

- What methods contained in an EJB's bean class can benefit from using the functionality found on a Data Access Object?

  Answer: Any method that accesses the entity data store can benefit from the functionality provided by a DAO, including finder and business methods, as well as any of a bean's life cycle methods that write to or read from the database, such as `ejbStore`, `ejbCreate`, and `ejbLoad`.

- What SQL commands are used to add a new row to a database table, remove a row from a database table, and modify one or more data elements in a row of a database table?

  Answer: INSERT adds a new row to a database table, DELETE removes a row from a database table, and UPDATE modifies one or more columns in a row of a database table.

- Describe how using a Data Access Object to encapsulate the bean's data access code affects the home and remote interface definitions.

  Answer: The data access implementation for the bean class methods should have no effect on the way they are defined on the home and remote interfaces.

- Discuss the design goals achieved by using private methods on the DAO to access the data store and not embedding the database access logic in the public methods invoked by the EJB.

  Answer: Using this design approach, the public methods exposed to the EJB remain constant regardless of the implementation of the data store access code.

- How does implementing an entity bean's data access code in a DAO impact an EJB client?

  Answer: The implementation of the bean class's data access code has no impact on an EJB client.

- How does implementing an entity bean's data access code in a DAO impact the steps taken when the bean is deployed?

  Answer: The DAO is a helper class. The class file for the DAO must be included in the `.jar` file that contains the EJB.

Sun Microsystems, Inc. Copyright 2001 All Rights Reserved. Enterprise Services, Revision C

Filename:	lab05.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 10:14 AM
Change Number:	2
Last Saved On:	04/24/01 10:14 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:	3,961 (approx.)
Number of Characters:	22,578 (approx.)