Exercise 6:

Creating Finder Methods for BMP Entity EJB™

Case Study: Creating Finder Methods for BMP Entity EJBs

Estimated completion time: 1 hour.

In this lab exercise, you construct special finder methods for the `Customer` and `Account` BMP entity EJB™ to retrieve all customers from the customer table and a set of accounts from the account table associated with a specified customer.

Figure 1 illustrates the updated `Customer` EJB. The `BankMgr` session EJB provides the access mechanism to the functionality contained on the `Customer` EJB. The `BankMgr`'s `getAllCustomers` method returns a `Vector` of `CustomerData` objects and invokes two methods on the `Customer` EJB to perform its duties. The `BankMgr` first calls `findAll` to retrieve a collection of remote references for the customers contained in the `J2EECUSTOMER` table and then creates a `Vector` of `CustomerData` objects representing the suite of customers using the `Customer` EJB's `getCustomerData` method for each customer instance. Retrieving all of the rows from a database table can be very resource-intensive and place a heavy burden on the EJB container.

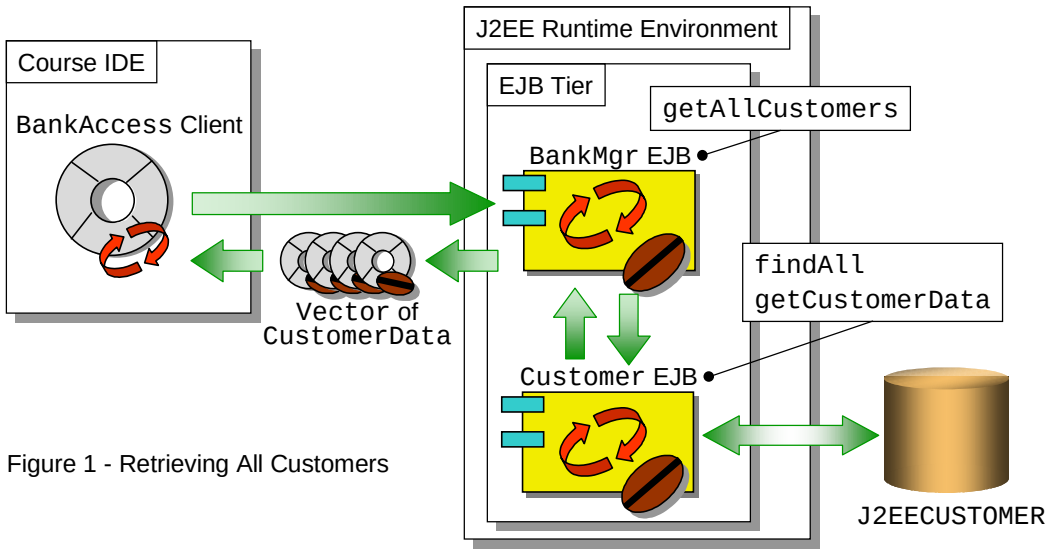

Figure 1 - Retrieving All Customers

Figure 2 illustrates the modified `Account` EJB. Notice that the `Account` EJB's `findByCustomerID` method returns a `Collection` of `Account` remote references. To retrieve the data for each `Account` instance, the client must invoke the `getAccountData` method on each instance. A business method on the `BankMgr` session bean performed this additional work when retrieving all customers from the customer table.

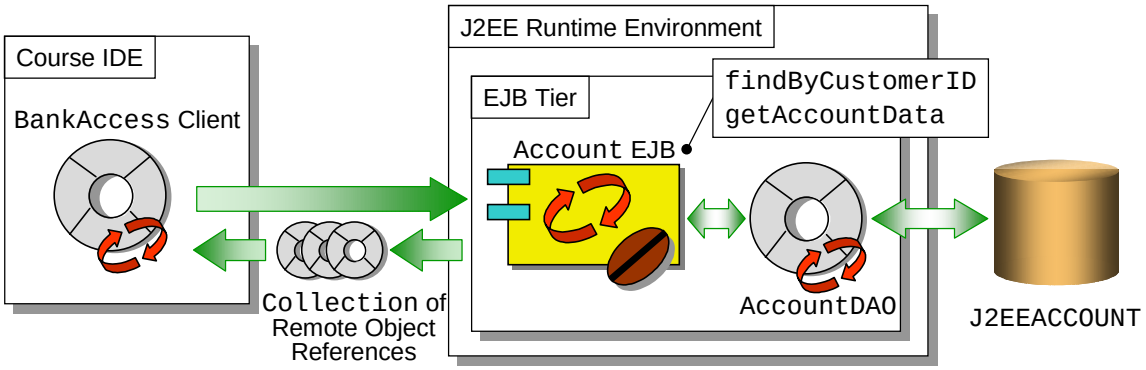

Figure 2 - Retrieving a Customer's Accounts

There are four sections to this lab exercise. In the first section, you code the `Customer` EJB's `findAll` method. In Section 2, you modify the `BankAccess` test client and `BankMgr` session EJB to invoke the new finder method and then deploy and test the application. In Section 3, you code the `Account` EJB's `findByCustomerID` method. Finally, in Section 4, you modify the `BankAccess` test client to invoke the `Account` EJB's new finder method and then deploy and test the application.

After completing this lab, you will be able to:

- Create a custom finder method for a basic BMP entity bean
- Create a custom finder method for a BMP entity EJB using a Data Access Object
- Describe the difference between single- and multi-object finder methods

Discussion Topics:

- What type of object is contained in the `Collection` returned by the bean class implementation of a multi-object finder method?
- What type of object is returned to the client application that invokes a multi-object finder method on a BMP entity EJB? Explain.
- How do the return types for multi-object finder methods differ from those for single-object finder methods?

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.sql
   public interface Statement
      public ResultSet executeQuery(String sql) throws SQLException
      public void close() throws SQLException
   public interface ResultSet
      public boolean next() throws SQLException
      public void close() throws SQLException
      String getString(int columnIndex) or String getString(String
                                    columnName)
   public interface Connection
      public Statement createStatement() throws SQLException
      public void close() throws SQLException
java.util
   public interface Collection
      public boolean add(Object o)
      public boolean isEmpty()
      public Iterator iterator()
   public class ArrayList extends AbstractList implements List,
                                    Cloneable, Serializable
      public boolean add(Object o)
javax.ejb
   public class EJBException extends java.lang.RuntimeException
   public class FinderException extends java.lang.Exception
   public class ObjectNotFoundException extends FinderException
   public interface EntityContext extends EJBContext
      java.lang.Object getPrimaryKey()
javax.sql
   public interface DataSource
      java.sql.Connection getConnection() throws java.sql.SQLException
```

Setup Instructions

This lab exercise requires no special setup. Modify the existing application files for the `Account` and `Customer` EJBs and the client components when completing this exercise. You can view an example of the completed exercise in the “Solution Code” section of this lab module.

Section 1

Modify the `Customer` BMP Entity EJB

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

1. Add a method stub for the new finder method to the `Customer` EJB's home interface:
   - Creator/finder methods:
     - `findAll` – Returns a collection representing all of the customers in the customer table
2. Add a method to the `Customer` EJB's bean class that implements the new finder method:
   - Name: `ejbFindAll`
   - Visibility: `public`
   - Return type: `java.util.Collection`
   - Parameters: None
   - Functionality: Use the example provided by the other methods on the `Customer` EJB as a guideline in coding this method. The method performs a `SELECT *` against the `J2EECUSTOMER` table and returns an `ArrayList` of `customerID`s, one for each row in the `J2EECUSTOMER` table.

Compile

3. 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 implication of implementing a finder method on a BMP entity EJB that executes the SQL command "`SELECT * FROM <TABLENAME>`".

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 Collection of CustomerData models representing the customers contained in the J2EECUSTOMER table.

1. Modify the BankMgr session EJB's remote interface as follows:
   - Business methods:
     - `java.util.Vector getAllCustomers`
2. Perform the following modifications to the BankMgr EJB's bean class:
   - Add a method to the bean class as follows:
     - Name: `getAllCustomers`
     - Visibility: `public`
     - Returns: `java.util.Vector` (of `CustomerData` objects)
     - Parameters: None
     - Functionality: Code the method to invoke the `Customer` EJB's `findAll` method, and then create a `Vector` of `CustomerData` objects using the returned `Collection` of `Customer` remote references.

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 Collection of CustomerData objects representing the customers found in the J2EECUSTOMER table. You access the BankMgr EJB's getAllCustomers method directly from BankAccess.main.

3. Modify the BankAccess's main method to call BankMgr.getAllCustomers, and display the resulting Vector of CustomerData objects.

Compile

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

Deploy and Test

In this section, you update and re-deploy and test the `BankApp` application. Consult the "Assembly and Deployment Guidelines" for information on updating and re-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.

5. Using the J2EE assembly/deployment tool provided for this course, update and re-deploy the `BankApp` application 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:
    - 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 `CustomerBean` and set the transaction attribute for the new finder method to "required".
        - Update the method transaction attributes for the `BankMgrBean` and set the transaction attribute for `getAllCustomers` to "required".
        - 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.
6. 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.

Section 3

Modify the `Account` BMP Entity EJB

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

1. Add a method stub for the new finder method to the `Account` EJB's home interface:
   - Creator/finder methods:
     - `findByCustomerID(String customerID)` – Returns a collection representing all of the accounts for a specified customer
2. Add a method to the `Account` EJB's bean class that implements the new finder method:
   - Name: `ejbFindByCustomerID`
   - Visibility: `public`
   - Return Type: `java.util.Collection`
   - Parameters: `String customerID`
   - Functionality: Use the example provided by the other methods on the `Account` EJB as a guideline in coding this method. This method invokes the `AccountDAO`'s `findByCustomerID` method provided with the `AccountDAO` skeleton code.

Compile

3. 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:

- What type of exception is thrown by a multi-object finder method on the bean class when no rows are found in the database that match the search criteria?
- What must an EJB client application do to access a data element from a table row located using a multi-object finder method?

Section 4

Modify the `BankAccess` Test Client

The `BankAccess` bean serves as the test client for this exercise. The `BankAccess` bean interacts directly with the `Account` EJB.

1. Modify the `BankAccess`'s `main` method to call `AccountHome.findByCustomerID` for a specific customer and display the returned accounts.

Compile

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

Deploy and Test

In this section, you update and re-deploy and test the `BankApp` application. Consult the "Assembly and Deployment Guidelines" for information on updating and re-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.

3. Using the J2EE assembly/deployment tool provided for this course, update and re-deploy the `BankApp` application 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`.
        - Update the method transaction attributes for the `AccountBean`, and set the transaction attribute for the new finder method to "required".
        - 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 using the Cloudscape database supplied with the J2EE Reference Implementation.
4. 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;

   public java.util.Collection findAll()
      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.*;
import java.util.*;

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) { ... }

   public void unsetEntityContext() { ... }

   public void ejbActivate() { ... }

   public void ejbPassivate() { ... }

   public void ejbLoad() { ... }

   private void loadRow() throws SQLException { ... }

   public void ejbStore() { ... }

   private void storeRow() throws SQLException { ... }

   public Collection ejbFindAll() throws FinderException {
      System.out.println("Entering CustomerEJB.ejbFindAll()");
      try {
         Collection col = selectAll();
         if (col.isEmpty())
            System.out.println(“No customers found in ejbFindAll()”);
         return col;
      } catch (Exception ex) {
         throw new EJBException("CustomerEJB.ejbFindAll " +
                                ex.getMessage());
      }
   }

   private Collection selectAll() throws SQLException {
      System.out.println("Entering CustomerEJB.selectAll()");
      Connection dbConnection = initDBConnection();
      Statement stmt = (this.dbConnection).createStatement();
      String queryStr =
         "SELECT * FROM J2EECUSTOMER";
      System.out.println("queryString is: "+ queryStr);
      ResultSet rs = stmt.executeQuery(queryStr);
      ArrayList a = new ArrayList();
      while (rs.next()) {
         String customerid = rs.getString(1);
         a.add(customerid);
      }
      stmt.close();
      dbConnection.close();
      System.out.println("Leaving CustomerEJB.selectAll()");
      return a;
   }

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

   private boolean selectByPrimaryKey(String key)
      throws SQLException { ... }

// business methods
   public CustomerData getCustomerData() { ... }
```

```
   // utility method
   private Connection initDBConnection()
      throws SQLException { ... }
}
```

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;

   public java.util.Vector getAllCustomers()
      throws RemoteException;
}

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

// BankMgrEJB.java

package J2EEApp;

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 java.util.Vector getAllCustomers() {
      System.out.println("Entering BankMgrEJB.getAllCustomers()");
      java.util.Vector customerList = new java.util.Vector();
      try {
         java.util.Collection cl = myCustomerHome.findAll();
         if (cl == null) { customerList = null; }
         else {
            java.util.Iterator cli = cl.iterator();
            while ( cli.hasNext() ) {
               customerList.addElement
                  (((Customer)cli.next()).getCustomerData());
            }
         }
      } catch (Exception e) {
         System.out.println
            ("Error in BankMgrEJB.getAllCustomers(): " + e);
      }
      System.out.println("Leaving BankMgrEJB.getAllCustomers()");
      return customerList;
   }
```

```
   public BankMgrEJB() {  }

   public void ejbCreate() { ... }

   public void setSessionContext(SessionContext sc) { }

   public void ejbRemove() { }

   public void ejbActivate() { }

   public void ejbPassivate() { }

   public String loginUser(String pUsername, String pPassword) { ... }

   public CustomerData getCustomerData(String customerID) { ... }
}

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

// 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();
         if (ba.myBankMgrHome == null) ba.initMyBankMgrHome();
         BankMgr myBankMgr = ba.myBankMgrHome.create();
         java.util.Vector customers = myBankMgr.getAllCustomers();
         myBankMgr.remove();

         if (customers == null)
            System.out.println("customers is null: BankAccess.main()");
         else {
            java.util.Iterator cli = customers.iterator();
            System.out.println("The list of customers is: ");
            while (cli.hasNext()) {
               System.out.println((CustomerData)cli.next());
            }
         }
      }
      catch (Exception e) {
         System.out.println("Error in BankAccess.main(): " + e);
      }
   }

    public void init() { ... }
```

```
    private void initMyBankMgrHome() { ... }

    private void initMyAccountHome() { ... }
}
```

Solution Code: Section 3

```
// AccountHome.java

package J2EEApp;

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

public interface AccountHome extends EJBHome {

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

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

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

}

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

// AccountEJB.java

package J2EEApp;

import java.sql.*;
import javax.sql.*;
import javax.ejb.*;
import javax.naming.*;
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 acctNumber, String customerID,
                            String description, double balance)
      throws DuplicateKeyException,CreateException { ... }

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

   public void ejbLoad() { ... }

   public void ejbStore() { ... }
```

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

   public Collection ejbFindByCustomerID (String customerID)
      throws FinderException {
      System.out.println("Entering AccountEJB.ejbFindByCustomerID()");
      AccountDAO ADAO = new AccountDAO();
      ADAO.setCustomerID(customerID);
      Collection col = null;
      try{
         Connection dbConnection = getDBConnection();
         col = ADAO.findByCustomerID(dbConnection);
         dbConnection.close();
      } catch (java.sql.SQLException se) {
         throw new FinderException
                     ("SQL Exception in find by customerID");
      }
      System.out.println("Leaving AccountEJB.ejbFindByCustomerID()");
      return col;
   }

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

   public void unsetEntityContext() { ... }

   public void ejbActivate() { ... }

   public void ejbPassivate() { ... }

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

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

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


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

// 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(){ ... }
public void setAcctNumber(String acctNumber){ ... }

public String getCustomerID(){ ... }
public void setCustomerID(String customerID){ ... }

public String getDescription(){... }
public void setDescription(String description){ ... }

public double getBalance(){... }
public void setBalance(double balance){ ... }

// public methods
public Collection findByCustomerID(Connection con)
   throws SQLException {
   dbConnection = con;
   try {
      Collection result = selectByCustomerID();
      if (result.isEmpty()) {
         throw new ObjectNotFoundException("No rows found.");
      }
      else return result;
   } catch (Exception ex) {
      throw new EJBException
                  ("ejbFindByLastName " + ex.getMessage());
   }
}

public void load ( Connection con )
   throws SQLException { ... }

public void store( Connection con )
   throws SQLException { ... }

public void remove( Connection con )
   throws SQLException { ... }

public void create(Connection con)
   throws SQLException, DuplicateKeyException { ... }

public String findByPrimaryKey(Connection con)
   throws SQLException { ... }

// private methods
private Collection selectByCustomerID() throws SQLException {
   System.out.println("Entering AccountDAO.selectByCustomerID()");
   Statement stmt = dbConnection.createStatement();
   String queryStr =
      "SELECT acctnumber FROM J2EEACCOUNT WHERE customerid = "
      + "'" + this.customerID + "'";
```

```
      System.out.println("queryString is: "+ queryStr);
      ResultSet rs = stmt.executeQuery(queryStr);
      ArrayList a = new ArrayList();
      while (rs.next()) {
         String acctnumber = rs.getString(1);
         a.add(acctnumber);
      }
      stmt.close();
      System.out.println("Leaving AccountDAO.selectByCustomerID()");
      return a;
   }

   private boolean accountExists (String pAccountID)
      throws SQLException { ... }

   private void selectAccount () throws SQLException { ... }

   private void insertAccount()
      throws SQLException,DuplicateKeyException { ... }

   private void deleteAccount() throws SQLException { ... }

   private void updateAccount() throws SQLException { ... }
}
```

Solution Code: Section 4

// 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");
         java.util.Collection al =
            ba.myAccountHome.findByCustomerID(myID);
         if (al == null) {
            System.out.println
               ("No accounts were found for customerID: " + myID );
         } else {
            java.util.Iterator ali = al.iterator();
            System.out.println("The list of accounts for customerID " +
                               myID + " is: ");
            while ( ali.hasNext() ) {
               System.out.println
                  (((Account)ali.next()).getAccountData());
            }
         }
         java.util.Vector customers = myBankMgr.getAllCustomers();
         myBankMgr.remove();
         if (customers == null)
            System.out.println
               ("customers is null in BankAccess.main()");
         else {
            java.util.Iterator cli = customers.iterator();
            System.out.println("The list of customers is: ");
            while (cli.hasNext()) {
               System.out.println((CustomerData)cli.next());
            }
         }
      } 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 type of object is contained in the `Collection` returned by the bean class implementation of a multi-object finder method?

  Answer: The bean class implementation of a multi-object finder method returns a collection of the bean's primary key class.

- What type of object is contained in the `Collection` returned to the client application when invoking a multi-object finder method on a BMP EJB? Explain.

  Answer: A client application receives a `Collection` of remote interface references to EJB instances. The container converts the `Collection` of primary key objects created by the bean class implementation to the corresponding EJB remote interface references.

- How do the return types for multi-object finder methods differ from those for single object finder methods?

  Answer: Multi-object finder methods return a collection of `Objects` whereas single-object finder methods return a single object.

- Discuss the implication of implementing a finder method on a BMP entity EJB that executes the SQL command "`SELECT * FROM <TABLENAME>`".

  Answer: Selecting all of the rows from a database table can place a tremendous strain on the EJB container.

- What type of exception is thrown by a multi-object finder method on the bean class when no rows are found in the database that match the search criteria?

  Answer: Single-object finder methods throw an `ObjectNotFoundException` when the requested entity object is not found. Multi-object finder methods do not throw an exception but instead return an empty collection.

- What must an EJB client application do to access a data element from a table row located using a multi-object finder method?

  Answer: An EJB client uses the remote interface reference returned to it to access the EJB methods as required.

Filename:	lab06.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 10:15 AM
Change Number:	2
Last Saved On:	04/24/01 10:15 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:	22
Number of Words:	3,882 (approx.)
Number of Characters:	22,128 (approx.)