Building WebSphereBank: Introduction to Enterprise Java Beans

What this exercise is about

This exercise will highlight how to build a J2EE 1.4 based application using IBM Rational Application Developer V6.0. The application you will build in this exercise will demonstrate several key concepts associated with the EJB specification. Specifically, you will become more familiar with Container-Managed Persistence (CMP) entity EJBs and using the EJB Query Language to retrieve business data.

Lab requirements

List of system and software required for the student to complete the lab.

- IBM Rational Application Developer V6.0 beta with embedded WebSphere Application Server V6.0 Test Environment installed with beta patches installed
- Lab source files (Labfiles60.zip) must be extracted to the root directory (i.e., C:\)

What you should be able to do

At the end of this lab you should be able to use IBM Rational Application Developer V6.0 to:

- Create enterprise beans compliant to the EJB 2.1 specification
- Configure a relationship between two CMP Entity EJBs
- Define EJB QL statements
- Import a WAR file and add the Web Project to an existing EAR Project
- Export an EAR file

Introduction

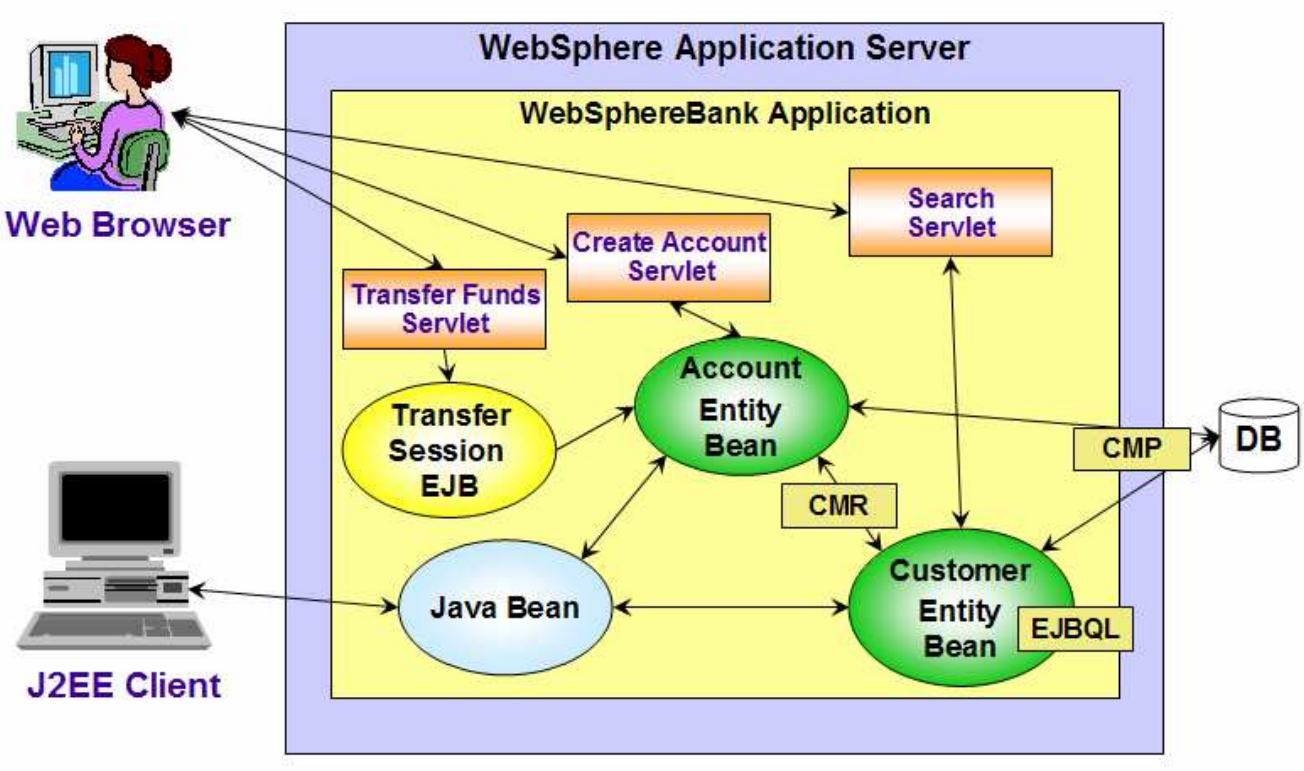
You will begin this exercise by building a small banking application which features CMP with IBM Rational Application Developer V6.0. You will start by creating an entity bean with Container-Managed Persistence which represents a bank account.

With the Account bean created, you will create a session enterprise bean which will use the local interface of the entity bean to perform a transfer of funds between accounts. The Transfer session bean will be generated with remote and local interfaces.

You will also create a second CMP EJB that will represent a customer in your banking application. The Customer and Account beans are associated through a Container Managed Relationship. This relationship is a One-To-Many unidirectional relationship. This means that a Customer bean will include behavior to access the Accounts that are owned by that Customer. You will also add several query functions to the Customer EJB using EJB QL.

With the business logic of your application created, you will next add a Web module and an application client module to your J2EE 1.4 application. The Web module contains several servlets which use the local interface of the session bean to transfer funds, while the application client will use the remote interface to perform the same transfer operation. With your J2EE 1.4 application complete, you will export the application as an Enterprise Archive (EAR) file.

The following diagram highlights the WebSphereBank Application.



Exercise instructions

Because these instructions are not operating-system specific, the directory locations will be specified in the lab instructions using symbolic references, as follows:

Reference variable	Windows® location	AIX®/UNIX® location
<LAB_FILES>	C:\Labfiles60	/tmp/Labfiles60

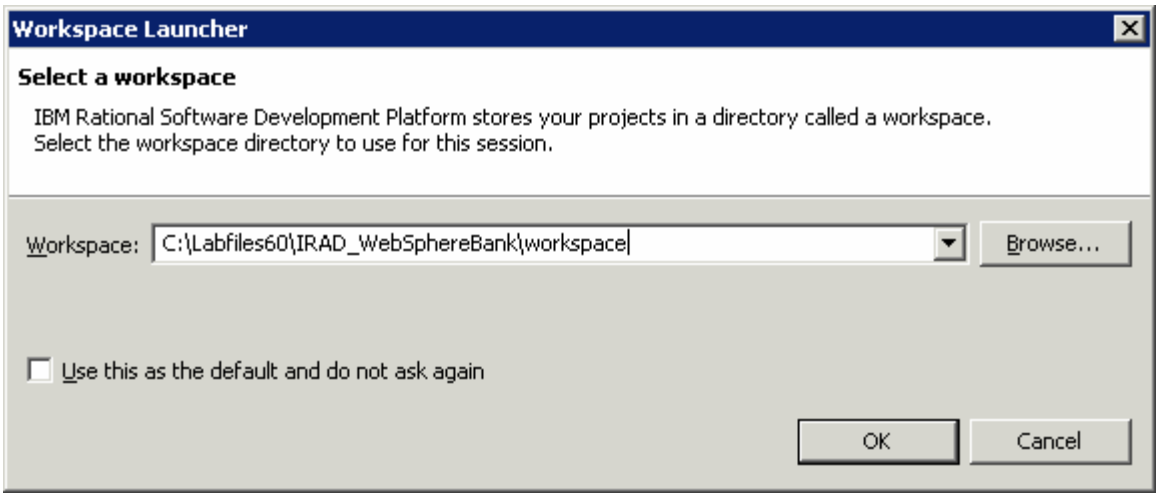
The following sections list the instructions for this lab.

Part 1: Setup development environment

____ 1. Start IBM Rational Application Developer V6.0.

__ a. Select **Start > Programs > IBM Rational > IBM Rational Application Development 6.0 > Rational Application Developer**.

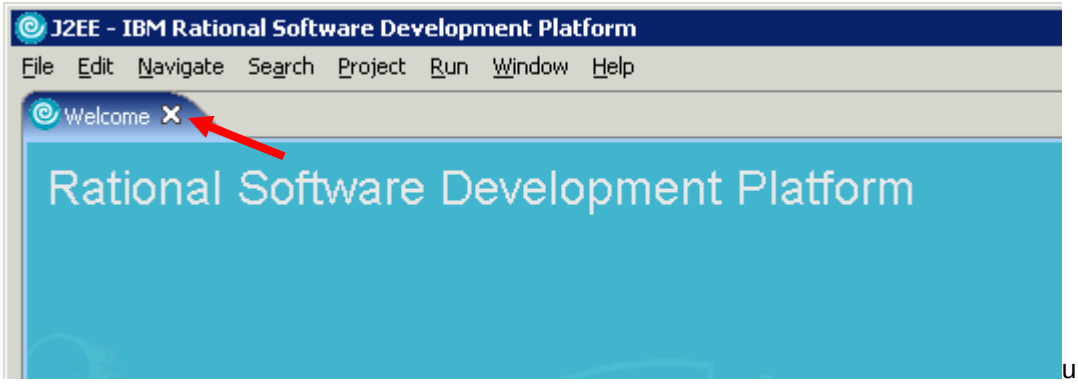
__ b. When prompted enter **<LAB_FILES>\IRAD_WebSphereBank\workspace** for your workspace.



__ c. Click **OK**.

NOTE: If the Auto Launch Configuration Change Alert window appears click the **Yes** button to change the auto launch eclipse instance to use when opening IBM Rational Software Development Platform in the future.

____ 2. When IBM Rational Application Developer V6.0 opens, click the **X** at the top left corner of the Welcome page.



u

Part 2: Build the WebSphereBank Enterprise application project

____ 1. Create a new J2EE Enterprise Application Project.

__ a. From the menu select **File > New > Enterprise Application Project**.

__ b. If the **Confirm Enablement** message dialog appears, click **OK**.

__ c. From the New Enterprise Application Project wizard, type **WebSphereBank** for the project name.

__ d. Click the **Show Advanced** button, and verify that the J2EE version is 1.4 and that the target application server is WebSphere Application Server V6.0. Click **Next**.

__ e. On the **EAR Module Projects** page, click the **New Module…** button.

__ f. Uncheck all other check boxes except for **Create default module projects** and **EJB Project**.

__ g. Click **Finish** to create the EJB Project module.

__ h. Click **Finish** to create the New Enterprise Application Project.

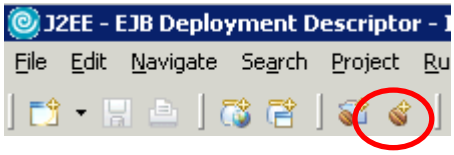
____ 2. Verify the Enterprise Application Project and EJB Project is visible from the Project Explorer view.

__ a. From the Project Explorer view you should see the **WebSphereBank** enterprise application project listed under Enterprise Applications and **WebSphereBankEJB** project listed under EJB Projects.

Part 3: Adding Enterprise Java Beans to WebSphereBank

____ 1. Create the Account entity bean.

__ a. From the toolbar click the **Create Enterprise Bean** button.



NOTE: Alternatively, you can select **File > New > Enterprise Bean**

__ b. Select the radio button next to **Entity bean with container-managed…**

__ c. Enter **Account** for the Bean name.

__ d. Enter **com.ibm.websphere.samples.bank.ejb** for the Default package field and click **Next**.

__ e. On the Enterprise Bean Details page check the box next to **Remote client view**.

NOTE: Providing a remote interface for entity beans is not a best practice. For this simple application however, a remote interface allows for easy testing of the application persistence from an external client.

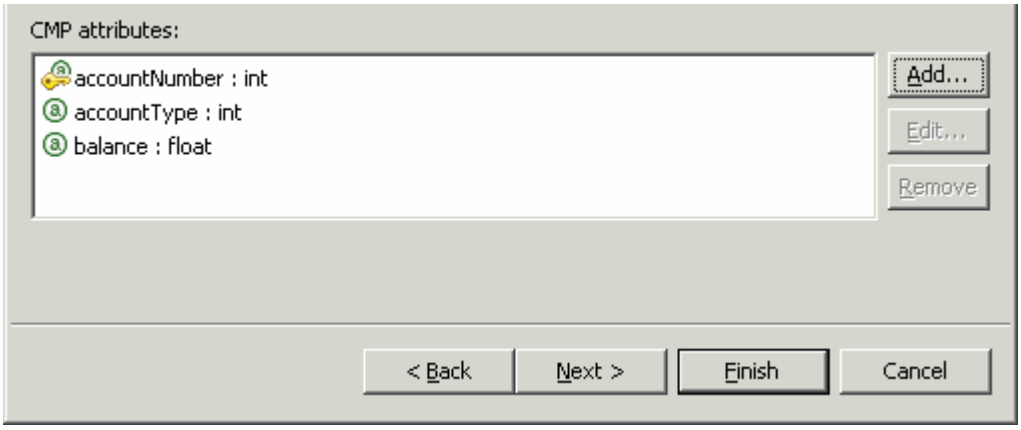
__ f. Select the **id:java.lang.Integer** entry in the list of CMP attributes and click **Remove**.

__ g. Click the **Add** button next the CMP attributes list.

__ h. Create the following 3 CMP attributes for the Account bean. To add an individual CMP attribute click **Apply**, and continue to the next attribute. When all of the attributes have been added click **Close**.

Name	Type	Key (select Key Field check box)
accountNumber	int	YES
accountType	int	NO
balance	float	NO

NOTE: Make sure to check the box next to **Promote getter and setter methods to remote interface** and **Promote getter and setter methods to local interface** for the non-key fields. Also, for the key field check the box next to **Key field**.

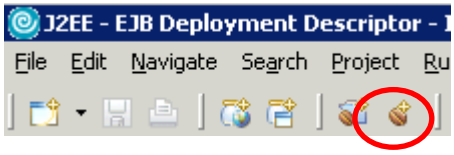


__ i. Click **Next** until you reach the **Select Class Diagram for Visualization** page, and uncheck the **Add bean to class diagram** check box.

__ j. Click **Finish** to create the Account entity bean.

____ 2. Create the Customer entity bean.

__ a. From the toolbar click the **Create Enterprise Bean** button.



NOTE: Alternatively, you can select **File > New > Enterprise Bean**.

__ b. Select the radio button next to **Entity bean with container-managed…**

__ c. Enter **Customer** for the Bean name.

__ d. Enter **com.ibm.websphere.samples.bank.ejb** for the Default package field and click **Next**.

__ e. On the Enterprise Bean Details page check the box next to **Remote client view**.

__ f. Select the **id:java.lang.Integer** entry in the list of CMP attributes and click **Remove**.

__ g. Click the **Add** button next the CMP attributes list.

__ h. Create the following 4 CMP attributes for the Customer bean. To add an individual CMP attribute click **Apply**, and continue to the next attribute. When all of the attributes have been added click **Close**.

Name	Type	Key (select Key Field check box)
customerNumber	long	YES
lastName	java.lang.String	NO
firstName	java.lang.String	NO
taxID	java.lang.String	NO

NOTE: Make sure to check the box next to **Promote getter and setter methods to remote interface** and **Promote getter and setter methods to local interface** for the non-key fields. Also, for the key field check the box next to **Key field**.

__ i. Click **Next** until you reach the "Select Class Diagram for Visualization" page, and uncheck the **Add bean to class diagram** check box.

__ j. Click **Finish** to create the Customer entity bean.

____ 3. Create the Transfer Stateless Session Bean.

__ a. From the toolbar click the **Create Enterprise Bean** button. Alternatively, you can select **File > New > Enterprise Bean**.

__ b. Select the radio button next to **Session Bean**.

__ c. Enter **Transfer** for the Bean name.

__ d. Enter **com.ibm.websphere.samples.bank.ejb** for the Default package field.

__ e. Click **Next**.

__ f. Check the **Remote Client View** and **Local Client View** check boxes.

__ g. Click **Next** until you reach the **Select Class Diagram for Visualization** page, and uncheck the **Add bean to class diagram** check box.

__ h. Click **Finish** to create the Transfer bean.

Part 4: Configure container managed relationships

____ 1. Open the EJB Deployment Descriptor.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB**.

__ b. Right click on **Deployment Descriptor: WebSphereBankEJB** and select **Open**.

____ 2. Specify the JNDI name for the CMP connection factory binding.

__ a. From the deployment descriptor editor, select the **Overview** tab.

__ b. Underneath the section **JNDI-CMP Connection Factory Binding** enter **jdbc/Bank** for the JNDI name and verify that the container authorization type is **Per_Connection_Factory**.

____ 3. Define JNDI Names for the Account, Customer, and Transfer Enterprise Java Beans.

__ a. From the deployment descriptor editor select the **Bean** tab.

__ b. For each EJB, shown on left hand side of the page, specify the appropriate JNDI name as indicated in the table below. To enter the JNDI name, click on the appropriate bean and then enter the name in the JNDI name text field under the WebSphere Bindings section.

Bean	JNDI Name
Account	ejb/Bank/Account
Customer	ejb/Bank/Customer
Transfer	ejb/Bank/Transfer

____ 4. Specify EJB References for the Account, Customer, and Transfer beans. The following is a summary of the references that need to be configured.

Bean	References
Account	ejb/Customer
Customer	ejb/Account
Transfer	ejb/Account

__ a. From the EJB deployment descriptor editor select the **Reference** tab.

__ b. Select the **Account** bean from the list on the left side of the page and click **Add**.

__ c. Select the radio button next to **EJB reference** and click **Next**.

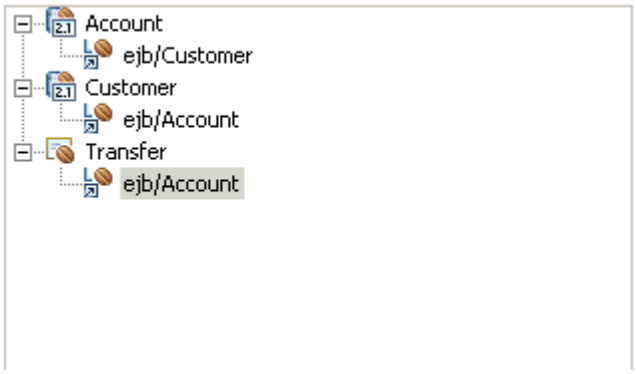
__ d. Expand **WebSphereBank > WebSphereBankEJB** and select **Customer**.

__ e. From the **Ref Type** drop down list select **Local**.

__ f. With the **Enterprise beans in the workspace** radio button selected, notice that the reference name is automatically filled in to be **ejb/Customer**.

__ g. Click **Next** and then **Finish**.

__ h. Repeat Steps 4b-g for the **Customer** and **Transfer** beans with the appropriate references. These beans both reference the **Account** bean.

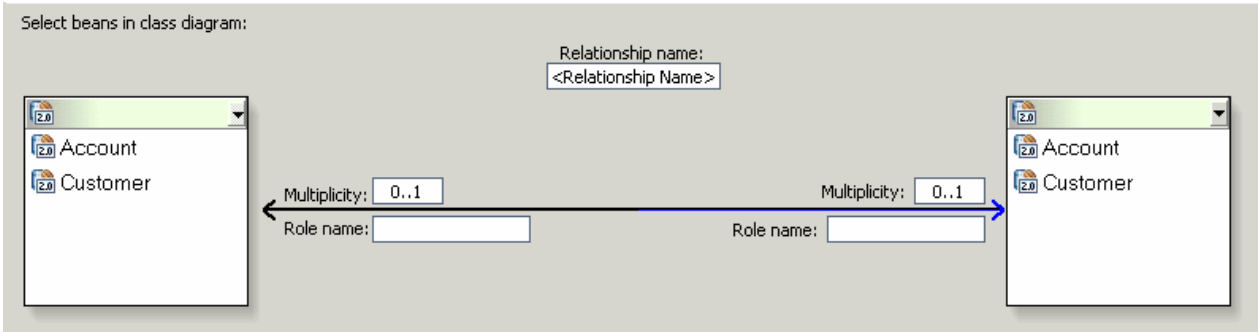


____ 5. Define the Container Managed Relationship between the Customer and the Account EJB.

__ a. From the EJB deployment descriptor editor select the **Overview** tab.

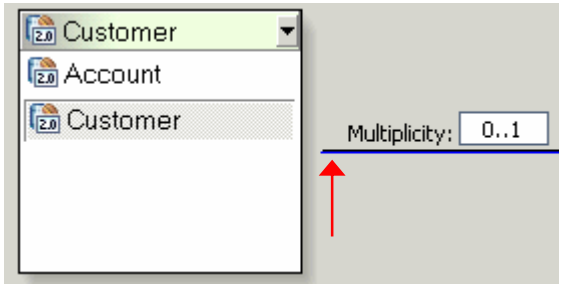
__ b. Scroll down until you see the section heading **Relationships 2.0** and click **Add**.

__ c. You will be presented with the following screen to build the relationship.



__ d. Select **Customer** from the list on the left.

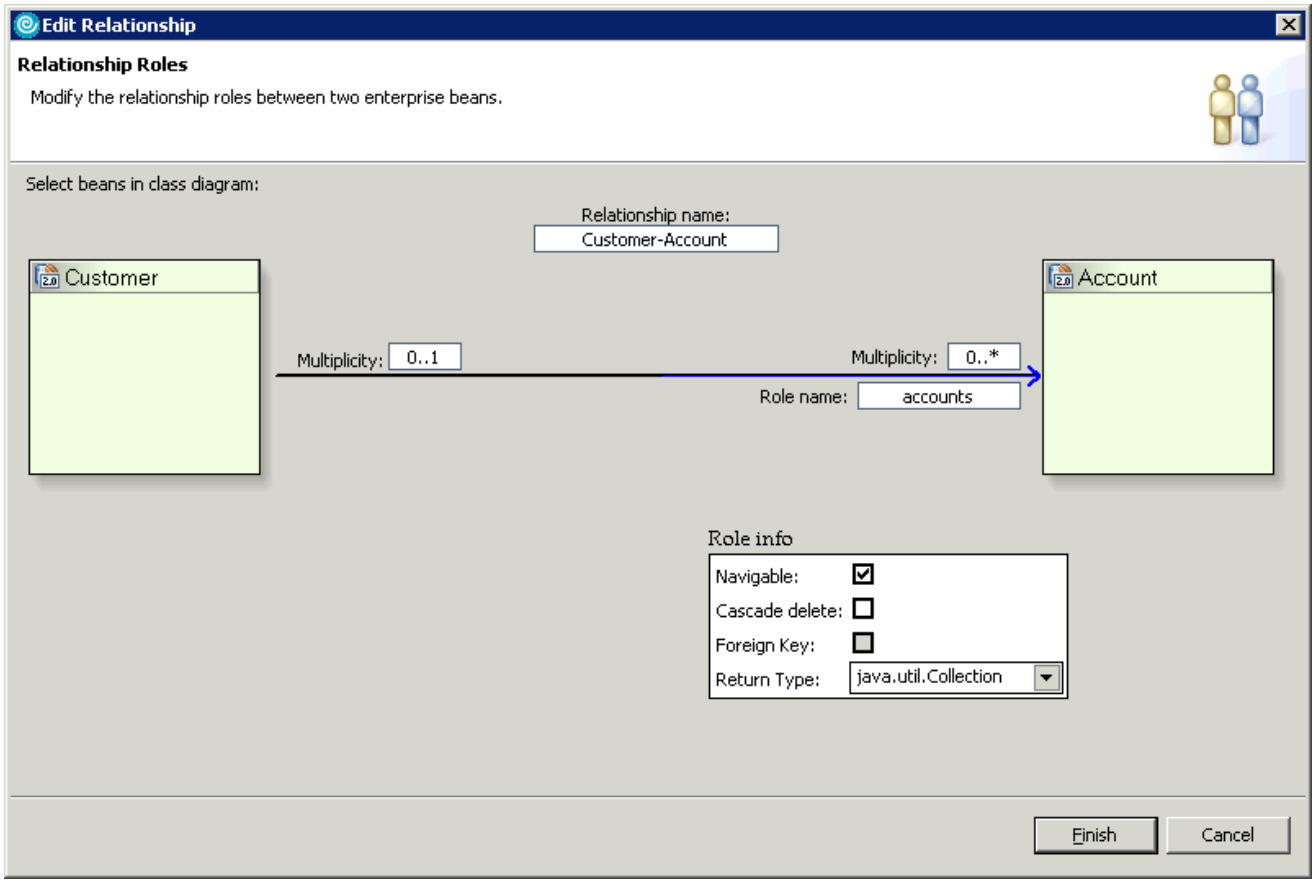
__ e. For the customer side of the relationship, verify that **Multiplicity** is 0..1, and make sure **customer** is specified as the **Role name**. Also, specify that this side of the relationship is not navigable. To do this, position your cursor as indicated by the arrow in illustration below. You will see a navigation box, and from it deselect **Navigable.**



__ f. In the box on the right, select **Account** from the drop down list. Change the **Multiplicity** to **0..***, and change the role name under Account to **accounts**.

__ g. To set the navigation for this side of the relationship, position your cursor on the arrow pointing to the **Account** box (as you did for the Customer side of the relationship). Ensure **Navigable** is selected, and verify that the Return Type is **java.util.Collection**.

__ h. Verify that your relationship is the same as the one below, and click **Finish**.



____ 6. Review the deployment descriptor source. Click on the **Source** tab. Locate the relationships definition and verify your code looks like the following.

```
<relationships>
    <ejb-relation>
        <ejb-relation-name>Customer-Account</ejb-relation-name>
        <ejb-relationship-role>
            <ejb-relationship-role-name>customer</ejb-relationship-role-name>
            <multiplicity>Many</multiplicity>
            <relationship-role-source>
                <ejb-name>Account</ejb-name>
            </relationship-role-source>
        </ejb-relationship-role>
        <ejb-relationship-role>
            <ejb-relationship-role-name>account</ejb-relationship-role-name>
            <multiplicity>One</multiplicity>
            <relationship-role-source>
                <ejb-name>Customer</ejb-name>
            </relationship-role-source>
            <cmr-field>
                <cmr-field-name>account</cmr-field-name>
                <cmr-field-type>java.util.Collection</cmr-field-type>
            </cmr-field>
        </ejb-relationship-role>
    </ejb-relation>
</relationships>
```

____ 7. **Save** and **Close** the EJB deployment descriptor.

____ 8. Verify that the appropriate methods have been added to the CustomerBean.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB > ejbModule > com.ibm.webspher.samples.bank.ejb,** and right click on the **CustomerBean.java file**. Select **Open** from the context menu.

__ b. From the outline view verify that the **getAccount()** and **setAccount(Collection)** methods have been added to the class definition.

__ c. **Close** CustomerBean.java.

Part 5: Adding business logic to your Enterprise Java Beans

____ 1. There are several classes that are referenced from within the business logic that are being added to the EJBs in this section. These class files have been provided for you with this lab in the BankUtilities.jar file. Import the BankUtilities.jar file into your EJB project.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB** and right click on the **ejbModule** folder. Select **Import** from the context menu.

__ b. From the import wizard, scroll to the bottom and select **Zip file** and click **Next**.

__ c. Click the **Browse** button near the top of the window, and navigate to **<LAB_FILES>\IRAD_WebSphereBank\BankUtilities.jar** and click **Open**.

__ d. Verify that you are importing into the **WebSphereBankEJB/ejbmodule** directory, and click **Finish**.

__ e. There will be some error messages generated in the **Problems** view. You can ignore these for now.

____ 2. Add business logic to the Account bean (AccountBean.java). There are several methods that will be added. First, there are two basic methods available for adding or subtracting funds from an account. In addition to this, the Account bean also implements the javax.ejb.TimedObject interface. You will also add several Timer methods used to support calculating interest for an account.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB > ejbModule > com.ibm.websphere.samples.bank.ejb** and right click on the AccountBean.java file. Select **Open** from the context menu.

__ b. Modify the AccountBean class to implement the javax.ejb.TimedObject interface. Your class definition should look like this.

```
public abstract class AccountBean implements javax.ejb.EntityBean,
javax.ejb.TimedObject
```

__ c. Add the **add(), subtract()** and **Timer** methods to the AccountBean class. The code for this is included in **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet1.txt**. Copy all of the code included in this file into the AccountBean class under the following line of code.

```
private javax.ejb.EntityContext myEntityCtx;
```

__ d. The following is a summary of the methods you will be adding.

Review the code for the add() and subtract() methods.

```
/**
public float add(float amount) {
    setBalance(getBalance() + amount);
    return getBalance();
}
/**
public float subtract(float amount) throws InsufficientFundsException {
    if (getBalance() < amount) {
        throw new InsufficientFundsException(bundle.getString("insufficientFunds"));
    }
    setBalance(getBalance() - amount);
    return getBalance();
}
```

Review the code for the Timer methods

```
public void myCreateTimer(double percent) throws javax.ejb.EJBException {
    TimerService timerService = null;
    try {
        timerService = getEntityContext().getTimerService();
    } catch (Exception e) {
        e.printStackTrace();
        throw new javax.ejb.EJBException(e.getMessage());
    }
    if (timerService != null) {
        try {
            Timer timer = timerService.createTimer(10000, 10000, new Double(percent));
        } catch (Exception e) {
            e.printStackTrace();
            throw new javax.ejb.EJBException("Failed to create Timer.");
        }
    } else {
        throw new javax.ejb.EJBException("Failed to get Timer Service.");
    }

}

public void myCancelTimer() throws javax.ejb.EJBException {
    TimerService timerService = null;
    try {
        timerService = getEntityContext().getTimerService();
    } catch (Exception e) {
        e.printStackTrace();
        throw new javax.ejb.EJBException(e.getMessage());
    }
    if (timerService != null) {
        try {
            Timer timer = (Timer) timerService.getTimers().iterator().next();
            timer.cancel();
        } catch (Exception e) {
            e.printStackTrace();
            throw new javax.ejb.EJBException("Failed to cancel Timer.");
        }
    } else {
        throw new javax.ejb.EJBException("Failed to get Timer Service.");
    }
}

public void ejbTimeout(Timer timer) {
    try {
        double percent = ((Double) timer.getInfo()).doubleValue();
        percent = percent / 100.0;
        double currentbalance = getBalance();
        add((float) (currentbalance * percent));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

__ e. Add the following **ejbCreate** and **ejbPostCreate** methods to AccountBean.java just before the closing bracket for the class definition. For your convenience, this code can be found in **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet2.txt**.

```
/**
public com.ibm.websphere.samples.bank.ejb.AccountKey ejbCreate(int accountNumber,
        int type, float balance, long customerNumber) throws javax.ejb.CreateException {
    setAccountNumber(accountNumber);
    setAccountType(type);
    setBalance(balance);
    return null;
}


/**
public void ejbPostCreate(int accountNumber, int type,
        float initialBalance, long customerNumber) throws javax.ejb.CreateException {
   try {
      AccountLocal thisAccount = (AccountLocal) getEntityContext().getEJBLocalObject();
      InitialContext initCtx = new InitialContext();
      CustomerLocalHome customerHome = (CustomerLocalHome) initCtx.lookup(CUSTOMER_JNDI_NAME);

      CustomerLocal customer = customerHome.findByPrimaryKey(new CustomerKey(customerNumber));
      customer.addAccount(thisAccount); //add this account
   } catch (Exception e) {
      e.printStackTrace(); //for debugging
      throw new CreateException(e.getMessage()); //throws create exception - something went wrong
   }
}
```

__ f. At this point there may be a number of errors indicating that the appropriate import statements have not been included in the AccountBean class definition. An easy way to add these import statements is to right click anywhere in the editor area, and from the context menu choose **Source > Organize Imports**.

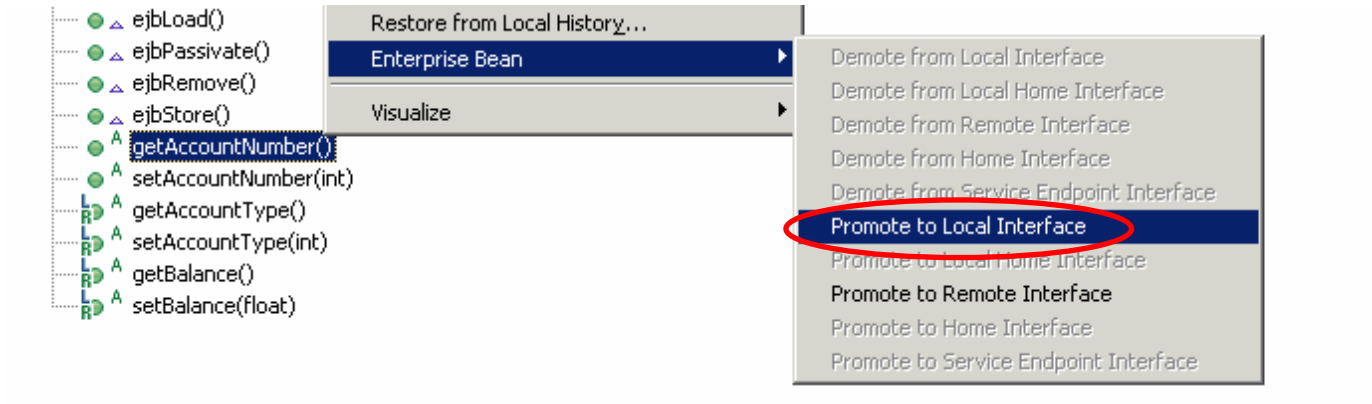
__ g. You will be prompted to choose the appropriate TimedObject class. Choose javax.ejb.Timer. You can also choose to format the code by pressing **Ctrl+Shift+F**. There will still be one remaining error in the AccountBean.java class concerning Customer.addAccount method, you can ignore this for now, and the error will be resolved later in this document.

__ h. Since you added the ejbCreate(int, int, float, long) method, you will need to promote this method to the Home and Local Home interface. In the Outline view, select **ejbCreate(int, int, float, long)** and right click. From the context menu, select **Enterprise Bean > Promote to Local Home Interface** and then **Enterprise Bean > Promote to Home Interface**.

__ i. The following table summarizes the methods you added earlier as well as the function getAccountNumber, and indicates which methods should be promoted to the Local and/or Remote interfaces. To promote a method to the Local or Remote interface, open the AccountBean.java class and right click on the appropriate method in the Outline view. From the context menu, select **Enterprise bean > Promote to Local Interface** and/or **Enterprise bean > Promote to Remote Interface** as appropriate. Since there are two methods choose **getAccountNumber()**.

Method	Local	Remote
add	YES	YES
subtract	YES	YES
myCreateTimer	YES	NO
myCancelTimer	YES	NO
getAccountNumber	YES	YES

__ j. The following screen capture highlights what you should see.

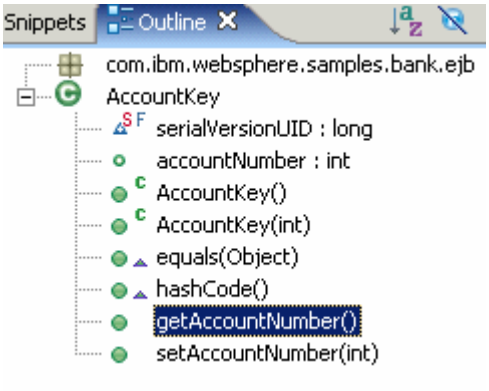


__ k. **Save** and **Close** the AccountBean.java file.

NOTE: There will still be errors in the Problem view that you will fix when you add the business logic for the Customer bean.

__ l. Open the **AccountKey.java** class.

__ m. Add get and set methods for the **accountNumber** member data. To do this, locate the line `public int accountNumber` and right click on it. From the context menu, select **Source > Generate Getters and Setters**. Make sure the check boxes next to the methods **getAccountNumber()** and **setAccountNumber(int)** are checked then click **OK**.



__ n. Verify that the appropriate methods have been added to the AccountKey class. **Save** and **Close** the AccountKey,java file.

____ 3. Add business logic to the Transfer bean (**TransferBean.java**). There are several methods you will add. First, there are two basic methods available for getting the balance for an account and transferring funds. In addition to this, there is another method that is used internally called **getAccountHome**.

__ a. Open the **TransferBean.java** file.

__ b. Add the code snippet included in the **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet3.txt** file to the top of the TransferBean class as indicated below by the arrow.

```
public class TransferBean implements javax.ejb.SessionBean {
    private javax.ejb.SessionContext mySessionCtx;
```

__ c. The accountHome member will need to be initialized with a handle to the Account bean. Add the following code to the ejbCreate method. For convenience, the code is provided in the **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet4.txt** file. Copy this code in between the open and close brackets of the ejbCreate method.

```
public void ejbCreate() {
    try {
        accountHome = getAccountHome();
    } catch (Exception e) {
        e.printStackTrace();
        throw new EJBException("Error getting accountHome: " + e.getMessage());
    }
}
```

__ d. Add the appropriate import statements. To do this, right click in the editor area and from the context menu choose **Source > Organize Imports**.

__ e. You will be prompted to choose the appropriate ObjectNotFoundException class. Choose javax.ejb.ObjectNotFoundException.

__ f. The methods you just added need to be made available to Local and Remote interfaces. To promote these methods to the Local and Remote interface, open the **TransferBean.java** class and right click on the appropriate method in the **Outline** view. From the context menu, select **Enterprise bean > Promote to Local Interface** and then **Enterprise bean > Promote to Remote Interface**.

Method	Local	Remote
getBalance	YES	YES
transferFunds	YES	YES

__ g. Save and **Close** the **TransferBean.java** file.

____ 4. Add business logic to the Customer bean (**CustomerBean.java**). There are three basic methods available: First, there is one for adding an account to the collection of accounts associated with a Customer (**addAccount**), another method is available for retrieving the Collection of owned accounts (**getOwnedAccountNumbers**), and finally a method that is used to retrieve the list of accounts owned by a particular Customer (**getAccountsList**). You will also add a new **ejbCreate** and **ejbPostCreate** method.

__ a. You will find the code that you are to add to the CustomerBean class in the **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet5.txt** file. This file includes the 4 methods discussed above. Add this code to the top of the CustomerBean class, right after the following line:

```
private javax.ejb.EntityContext myEntityCtx;
```

__ b. Review the **addAccount, getOwnedAccountNumber, and getAccountsList** methods shown below.

```
public void addAccount(AccountLocal anAccount) { //added for CMR
    if (anAccount != null)
        getAccounts().add(anAccount);
}

/*
public Vector getOwnedAccountNumbers() {
    Vector allAccountNumbers = new Vector();
    Collection allAccounts = getAccounts();
    Iterator iterator = allAccounts.iterator();
    while (iterator.hasNext()) {
        AccountLocal account = (AccountLocal) iterator.next();
        Long acctNumber = new Long(account.getAccountNumber());
        allAccountNumbers.add(acctNumber);
    }
    return allAccountNumbers;

}
```

```
    public Collection getAccountsList() {

        ArrayList list = null;
        try {
            Collection allAccounts = getAccounts();

            Iterator it = allAccounts.iterator();
            while (it.hasNext()) {
                if (list == null)
                    list = new ArrayList();

                AccountLocal account = (AccountLocal) it.next();
                long acctNumber = (new Long(account.getAccountNumber()))
                        .longValue();
                int acctType = account.getAccountType();
                float balance = account.getBalance();
                AccountData data = new AccountData(acctNumber, balance,
                        acctType);
                list.add(data);
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
        return list;
    }
```

__ c. Review the ejbCreate and ejbPostCreate methods shown below.

```
    public com.ibm.websphere.samples.bank.ejb.CustomerKey ejbCreate(
            CustomerKey customerKey, String name, String lname, String taxID)
            throws javax.ejb.CreateException {
        setCustomerNumber(customerKey.getCustomerNumber());
        setFirstName(name);
        setLastName(lname);
        setTaxID(taxID);
        return null;
    }

    /**
    public void ejbPostCreate(CustomerKey customerKey, String name,
            String lname, String taxID) throws javax.ejb.CreateException {
    }
```

__ d. Add the appropriate import statements. To do this, right click in the editor area and from the context menu choose **Source > Organize Imports**.

__ **e.** You will be prompted to choose the appropriate Iterator class. Choose **java.util.Iterator**. You can also choose to format the code by pressing **Ctrl+Shift+F**. Also note there may still be some errors indicated in the CustomerBean class, but these errors will be addressed in the following steps.

__ **f.** Since you added the **ejbCreate(CustomerKey, String, String, String)** method, you will need to promote this method to the Home and Local Home interface. In the **Outline view**, select **ejbCreate(CustomerKey, String, String, String)** and right click. From the context menu, select **Enterprise Bean > Promote to Local Home Interface** and then **Enterprise Bean > Promote to Home Interface**.

__ g. The following table summarizes the methods you added earlier, and indicates which methods should be promoted to the Local and/or Remote interfaces. To promote a method to the Local or Remote interface, open the **CustomerBean.java** class and right click on the appropriate method in the **Outline** view. From the context menu, select **Enterprise bean > Promote to Local Interface** and/or **Enterprise bean > Promote to Remote Interface** as appropriate.

Method	Local	Remote
getOwnedAccountNumbers	NO	YES
addAccount	YES	NO
getAccountsList	NO	YES

__ h. Save and Close the **CustomerBean.java** file.

__ i. Open the **CustomerKey.java** class.

__ j. Add get and set methods for the customerNumber member data. To do this, locate the following line of code.

```
public long customerNumber
```

Right click on this line and, from the context menu, select **Source > Generate Getters and Setters**. Make sure the check boxes next to the methods **getCustomerNumber** and **setCustomerNumber** are checked so they will be added to the class definition. Click **OK**

__ k. Verify that the appropriate methods have been added to the CustomerKey class. Save and Close the **CustomerKey,java** file.

NOTE: You should now only see 3 errors indicated in the Problems view. These errors will be addressed in the next section.

Part 6: Adding EJB QL statements

____ 1. Open the EJB Deployment Descriptor.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB**.

__ b. Right click on **Deployment Descriptor: WebSphereBankEJB** and select **Open**.

____ 2. Add an ejbSelect method to the CustomerBean that will select accounts by balance for a particular Customer.

__ a. Click on the **Bean** tab in the deployment descriptor editor, and select the **Customer** bean.

__ b. In the scroll area at the top right, scroll down to the Queries section and click **Add**.

__ c. Select **New** for the Method and **ejbSelect method** for the Method type. Enter **ejbSelectAccountsByBalance** for the name.

__ d. Click **Add** to add the following parameters.

Parameter	Type
customerNumber	long
balance	float

__ e. Enter **java.util.Collection** for the return type (or select from the list).

__ f. Click **Next** and enter the following EJB QL query in the Query statement text area (it can also be copied from the file **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet6.txt**):

```
SELECT a.accountNumber FROM Customer c, IN(c.accounts)a WHERE
c.customerNumber = ?1 AND a.balance > ?2
```

__ g. Click **Finish** to complete the creation of the ejbSelect statement.

____ 3. Add an ejbSelect method to the CustomerBean that will return the number of accounts owned by a particular customer.

__ a. While still in the Queries section of Customer on the Bean view of the EJB Deployment Descriptor, click **Add**.

__ b. Select **New** for the Method and **ejbSelect method** for the Method type. Enter **ejbSelectAccounts** for the name.

__ c. Click on **Add** to specify the following parameters.

Parameter	Type
customerNumber	Long

__ d. Enter **int** for the return type (or select from the list).

__ e. Click **Next** and enter the following EJB QL query in the Query statement text area (it can also be copied from the file **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet6.txt**):

```
SELECT COUNT(a) FROM Customer c, IN(c.accounts) a WHERE
c.customerNumber = ?1
```

__ f. Click **Finish** to complete the creation of the ejbSelect statement.

____ 4. Add a find method to the CustomerBean that will find the appropriate Customer object from a specified last name.

__ a. While still in the Queries section of Customer on the Bean view of the EJB Deployment Descriptor, click **Add**.

__ b. Select **New** for the Method and **find method** for the Method type. Enter **findByLastName** for the name.

__ c. For the Type field select both **Local** and **Remote**.

__ d. Click **Add** to define the following parameters.

Parameter	Type
name	java.lang.String

__ e. Enter **java.util.Collection** for the return type (or select from the list).

__ f. Click **Next** and enter the following EJB QL query in the Query statement text area (it can also be copied from the file **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet6.txt**):

```
SELECT OBJECT (c) FROM Customer c WHERE c.lastName LIKE ?1 ORDER BY
c.lastName
```

__ g. Click **Finish** to complete the creation of the find statement.

____ 5. Review the deployment descriptor code.

__ a. Click on the **Source** tab. Scroll down to the section where the Customer bean is defined. Notice the following lines of XML.

```
<query>
    <description></description>
    <query-method>
        <method-name>ejbSelectAccountsByBalance</method-name>
        <method-params>
            <method-param>long</method-param>
            <method-param>float</method-param>
        </method-params>
    </query-method>
    <ejb-ql>SELECT a.accountNumber FROM Customer c, IN(c.accounts) a WHERE c.customerNumber = ?1
            AND a.balance &gt; ?2</ejb-ql>
</query>
<query>
    <description></description>
    <query-method>
        <method-name>ejbSelectAccounts</method-name>
        <method-params>
            <method-param>long</method-param>
        </method-params>
    </query-method>
    <ejb-ql>SELECT COUNT(a) FROM Customer c, IN(c.accounts) a WHERE c.customerNumber = ?1</ejb-ql>
</query>
<query>
    <description></description>
    <query-method>
        <method-name>findByLastName</method-name>
        <method-params>
            <method-param>java.lang.String</method-param>
        </method-params>
    </query-method>
    <ejb-ql>SELECT OBJECT (c) FROM Customer c WHERE c.lastName LIKE ?1 ORDER BY c.lastName</ejb-ql>
</query>
```

____ 6. **Save** and **Close** the EJB Deployment Descriptor editor.

____ 7. Next examine the code created by Rational Application Developer to support the ejbSelectAccountsByBalance and ejbSelectAccounts EJB QL statements

__ a. From the Project Explorer view, navigate through **EJB Projects > WebSphereBankEJB > ejbModule > com.ibm.websphere.samples.bank.ejb** and open the file CustomerBean.java.

__ b. Scroll down to the section of the code where the method ejbSelectAccountsByBalance and ejbSelectAccounts are located. Notice that the select methods must be an abstract method, and that the code uses the naming prefix ejbSelect plus the method name

```
public abstract void setAccount(java.util.Collection anAccount);
public abstract java.util.Collection ejbSelectAccountsByBalance(long customerNumber, float balance) throws javax.ejb.FinderException;
public abstract int ejbSelectAccounts(long customerNumber) throws javax.ejb.FinderException;
```

____ 8. Because select methods cannot be directly exposed to the client, you will need to add two methods that expose the ejbSelect methods added earlier in this section. In this simple example, you need business logic methods that expose the results so that your Web module can use them.

__ a. **Open** the snippet of code provided in **<LAB_FILES>\IRAD_WebSphereBank\snippets\snippet6.txt**

__ b. Cut and paste the **getAccountsByBalance** and the **getAccountsCount** methods, included in this file, into the body of the CustomerBean class in the Java editor.

```
public Vector getAccountsByBalance(float minimumBalance) {
    try {
        Vector v = new Vector();
        Collection c = ejbSelectAccountsByBalance(getCustomerNumber(),
                minimumBalance);
        Iterator iterator = c.iterator();
        while (iterator.hasNext()) {
            v.add(iterator.next());
        }
        return v;
    } catch (javax.ejb.FinderException e) {
        e.printStackTrace();
    }
    return null;
}

public int getAccountsCount() {
    int c = 0;
    try {
        c = ejbSelectAccounts(getCustomerNumber());
    } catch (javax.ejb.FinderException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return c;
}
```

__ c. The methods you just added need to be made available to Local and/or Remote interfaces. To promote a method to the Local or Remote interface, open the **CustomerBean.java** class and right click on the appropriate method in the **Outline** view. From the context menu, select **Enterprise bean > Promote to Local Interface** or **Enterprise bean > Promote to Remote Interface** as appropriate. The following table summarizes this:

Method	Local	Remote
getAccountsByBalance	NO	YES
getAccountsCount	YES	YES

__ d. **Save** and **Close** the CustomerBean.java file.

Part 7: Import Web project and export WebSphereBank EAR

____ 1. Verify that there are no errors listed in the **Problems** view. You should only see one informational message listed.

____ 2. Import the WebSphereBank Web module.

__ a. Select **File > Import** from the menu.

__ b. Select **WAR File** from the list and click **Next**.

__ c. Click the Browse button and navigate to **<LAB_FILES>\IRAD_WebSphereBank\WebSphereBankWeb.war**. Click **Open**.

__ d. The Web Project should be set to **WebSphereBankWeb**, and select the EAR project: **WebSphereBankWebEAR** from the dropdown. Verify that the **Add module to an EAR project** check box is selected and click **Finish**.

__ e. If the **Confirm Perspective Switch** window appears, click **Yes**.

__ f. You should see a number of errors displayed in the Problems view. To resolve these errors, you need to update the Java Build Path for the Web module just imported. In the **Project Explorer** view, right click on the **WebSphereBankWeb** project and select **Properties** from the context menu. Click on **Java Build Path** and then click the **Projects** tab. Check the **WebSphereBankEJB** checkbox and click **OK**. The errors in the Problems view should disappear.

____ 3. Import the **FindAccounts** and **GetAccounts** Application Client modules.

__ a. Select **File > Import** from the menu.

__ b. Select **App Client JAR File** from the list and click **Next**.

__ c. Click the **Browse** button and navigate to **<LAB_FILES>\IRAD_WebSphereBank\FindAccounts.jar** and click **Open**.

__ d. The Application Client project should already be set to **FindAccounts**, the EAR project needs to be changed to **WebSphereBank**. Verify that the **Add module to an EAR project** check box is selected and click **Finish**.

__ e. If the **Confirm Perspective Switch** window appears, click **Yes**.

__ f. You should see a number of errors displayed in the Problems view. To resolve these errors, you need to update the Java Build Path for the Application Client module just imported. In the **Project Explorer** view expand **Application Client Projects**, and right click on the **FindAccounts** project and select **Properties** from the context menu. Click on **Java Build Path** and then click the **Projects** tab. Check the **WebSphereBankEJB** checkbox and click **OK**. The errors in the Problems view should disappear.

__ g. Repeat Steps 3a-f above for the **GetAccounts** application client module. For this module you will import the file: **<LAB_FILES>\IRAD_WebSphereBank\GetAccounts.jar**.

____ 4. Export WebSphereBank EAR file.

__ a. Click on **File > Export** and select **EAR File** from the list. Click **Next**.

__ b. Select **WebSphereBank** from the drop down menu for the Enterprise Application project.

__ c. Browse to **<LAB_FILES>\IRAD_WebSphereBank** and enter **WebSphereBank** for the name of the EAR file. Click **Save**.

__ d. Check the **Export source files** and **Include project build paths and meta-data files** check boxes and click **Finish**.

What you did in this exercise

In this exercise you built a simple J2EE 1.4 based banking application using IBM Rational Application Developer V6. This application highlights several important concepts associated with the EJB specification. Specifically, you gained experience with Container-Managed Persistence (CMP) entity EJBs and using the EJB Query Language statements to retrieve business data.

Solution Instructions

____ 1. Import the WebSphereBank application into Rational Application Developer for testing.

__ a. Select **File > Import...**

__ b. Select **EAR file** and select **Next**.

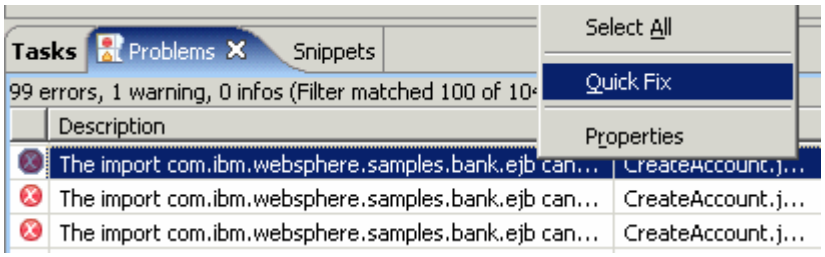
__ c. Select **Browse...** and navigate to **<LAB_FILES>\IRAD_WebSphereBank\solution\WebSphereBank.ear** and select **Open**.

__ d. For the Project name, enter **WebSphereBank**.

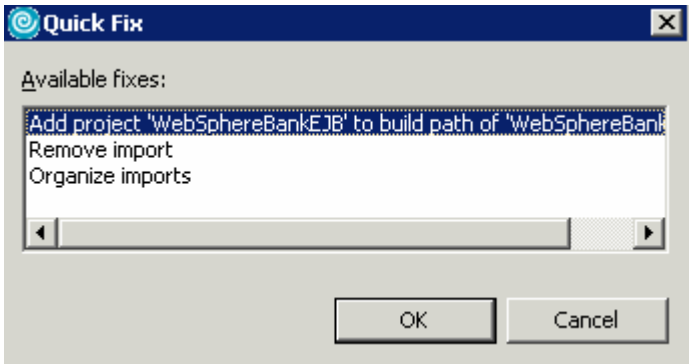
__ e. Click **Finish**.

____ 2. **Add Java Build Path** to clean up errors. If you select the Problems tab, you will notice a list of errors and warnings.

__ a. Right click on an error and select **Quick Fix.**



__ b. A list of available fixes will appear. Select **Add project 'WebSphereBankEJB' to build path of 'WebSphereBankWeb'** and select **OK.** The errors should disappear.



____ 3. Explore the WebSphereBankEJB project, and the various EJBs developed in this lab.

NOTE: The next lab in this series will test the functionality developed in this exercise for the WebSphereBank application.

Trademarks and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	iSeries	OS/400	Informix	WebSphere
IBM(logo)	pSeries	AIX	Cloudscape	MQSeries
e(logo)business	xSeries	CICS	DB2 Universal Database	DB2
Tivoli	zSeries	OS/390	IMS	Lotus

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Microsoft, Windows, Windows NT, and The Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds. Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.