IBM RATIONAL APPLICATION DEVELOPER 6.0 – LAB EXERCISE

Rapid Application Development with UML and Annotation Based Programming

What this exercise is about 1
Lab Requirements 2
What you should be able to do 2
Introduction 2
Exercise Instructions 3
Part 1: Setup Development Environment 4
Part 2: Planning and Designing Application 5
Part 3: Build the WebSphereBank Enterprise Application Project 5
Part 4: Adding Enterprise Java Beans to WebSphereBank 8
Part 5: Configure Container Managed Relationships 12
Part 6: Adding Business Logic to our Enterprise Java Beans 15
Part 7: Adding EJB QL Statements 24
Part 8: Complete Application 27
What you did in this exercise 29
Solution Instructions 30

What this exercise is about

This exercise will highlight how to build a J2EE 1.4 application using the rapid application development support provided with IBM Rational Application Developer v6.0. Applications which define business objects with entity beans can be designed visually using UML notations with the UML editor. The underlying Java artifacts will be created as the objects and relationships are created in the editor. Another feature available for rapid application development is annotation based programming. In J2EE artifacts, tags based on the proposed XDoclet specification may be used to define the different parts and specify deployment information in a single file. Developers can focus on defining the details and creating the business logic rather than managing all of the resources and searching for the correct location in the deployment descriptor to specify the correct information. The application you will build in this exercise, using UML and annotations, will demonstrate several key concepts associated with the Enterprise JavaBean 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 with embedded WebSphere Application Server v6.0 Test Environment 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 with the UML editor
- Configure a relationship between two CMP Entity EJBs with the UML editor
- Define EJB QL statements using annotations
- 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 Container-Managed Persistence (CMP) with IBM Rational Application Developer v6.0. Using the UML editor, you will start by creating an entity bean with Container-Managed Persistence which represents a bank account. The UML editor provides a UML view into the contents of the account entity bean. When artifacts are created in the UML editor, the Java artifacts are created. During creation, annotation tags can be used in defining the different parts of the Enterprise JavaBean.

You will also create a second CMP EJB that will represent a customer in the banking application. The Customer and Account beans are associated through a Container Managed Relationship. This relationship is a One-To-Many unidirectional relationship where the Customer bean will include behavior to access the Accounts that are owned by that Customer. The relationship can also be established using the UML editor.

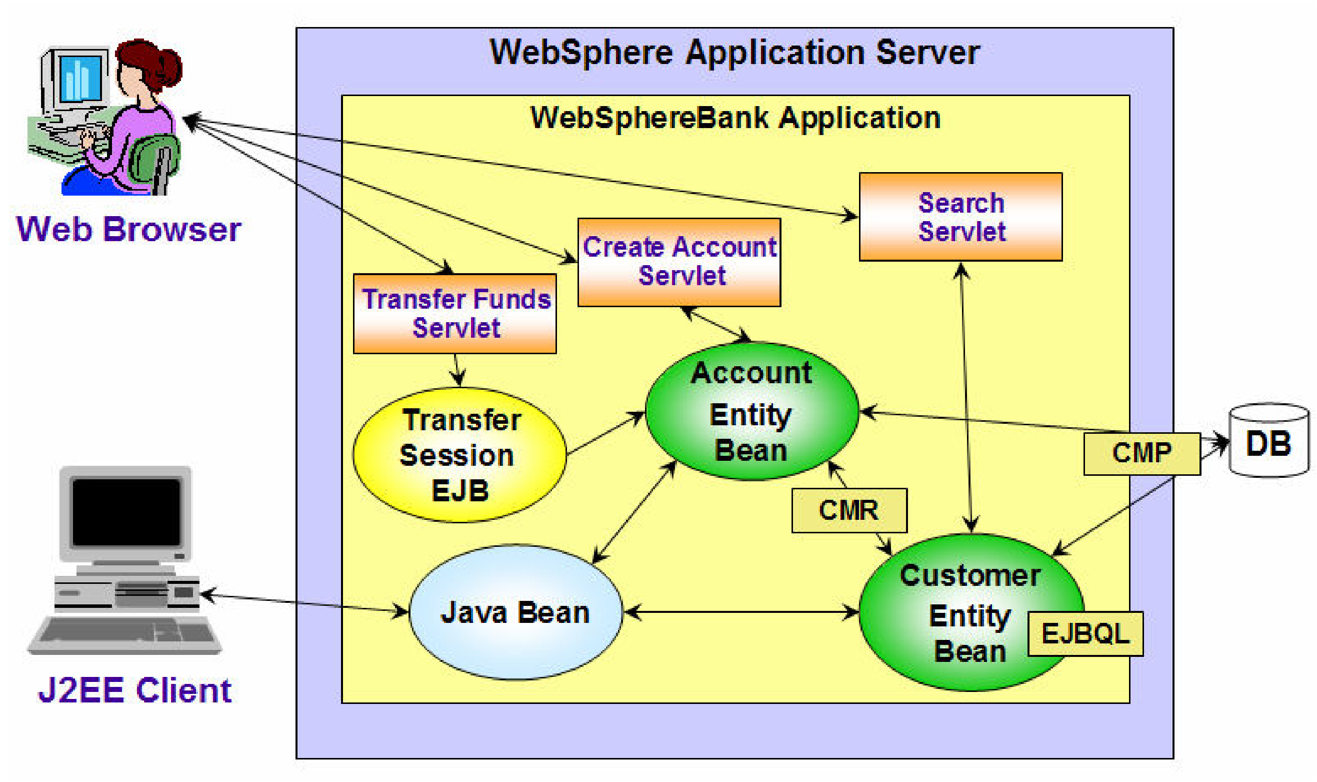
With the Account bean created, you will create a session enterprise bean which will use the local interface of the entity bean to perform a transfer of funds between accounts. The Transfer session bean will be generated with remote and local interfaces.

References between the different beans will also be defined through the UML editor.

After defining the references, you will then define the business logic by adding the appropriate Java methods. To promote these methods to the correct local or remote interfaces, annotation tags can be used. You will also add several query functions to the Customer EJB using EJB QL with annotation tags.

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
<RAD_HOME>	C:\Program Files\IBM\Rational\SDP\6.0	

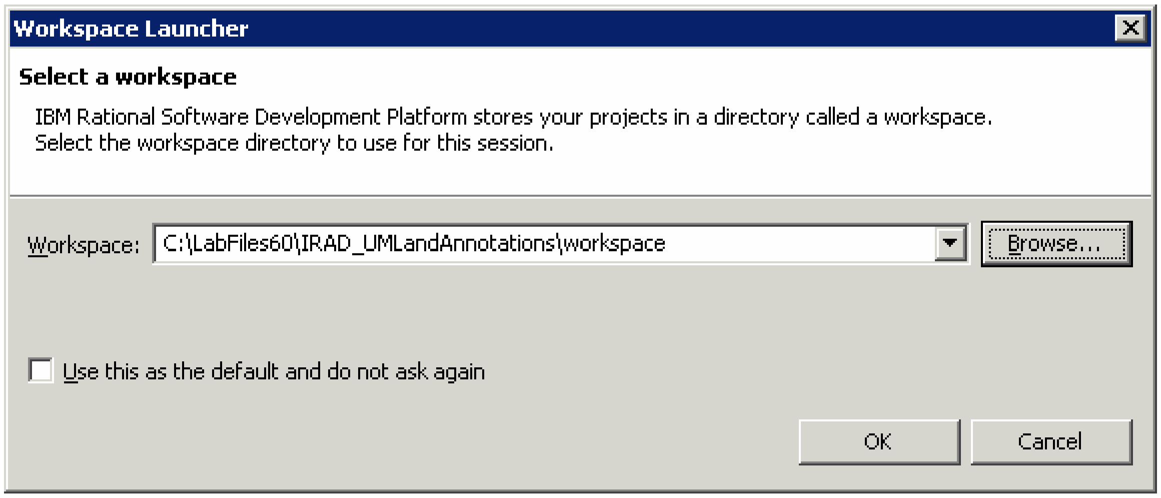
The following sections list the instructions for this lab.

Part 1: Setup Development Environment

____ 1. Start IBM Rational Application Developer v6.0.

__ a. Select **Start** > **Programs** > **IBM Rational** > **IBM Rational Application Development V6.0** > **Rational Application Developer**.

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



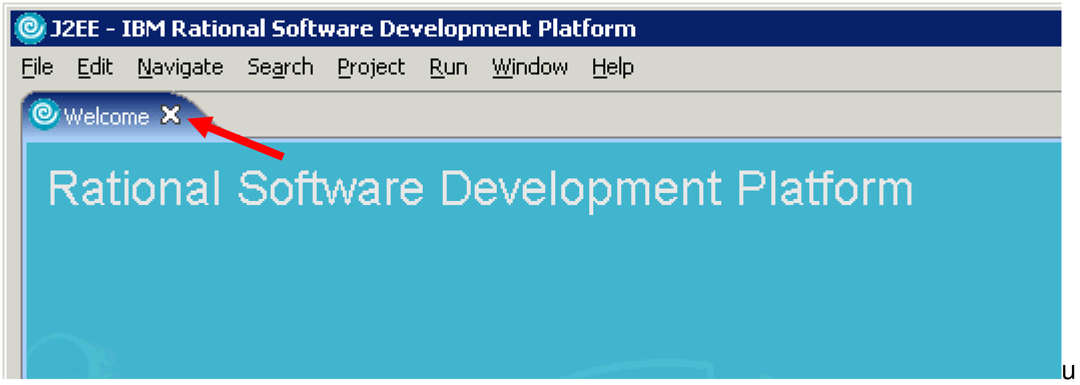
IBM Rational Software Development Platform stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\LabFiles60\IRAD_UMLandAnnotations\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.


J2EE - IBM Rational Software Development Platform
File Edit Navigate Search Project Run Window Help
Welcome
Rational Software Development Platform

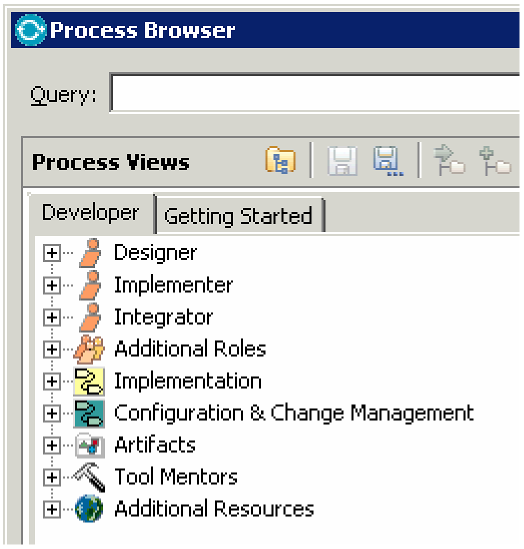
Part 2: Planning and Designing the Application

Before starting any enterprise application, careful planning and thorough design should be performed. There are many strategies and design patterns for building applications. One set of guidelines for application development is the Rational Unified Process (RUP). RUP is a software engineering process that provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget. IBM Rational Application Developer includes a library of RUP information which developers can conveniently access as they develop J2EE enterprise applications.

____ 1. View the Rational Unified Process information included with IBM Rational Application Developer

__ a. Select **Help > Process Browser**.

__ b. The different topics for Getting Started will be displayed. Select the Developer tag, to view the different topics organized by different roles.



__ c. Click the **Scope** button, in this dialog, you can filter what content searches should include.

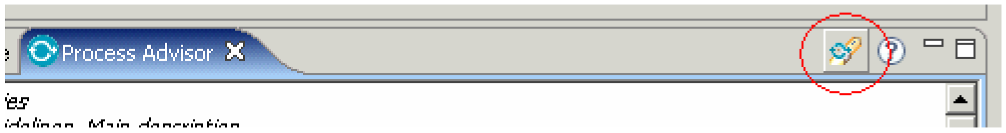
__ d. Close the Process Browser window.

____ 2. The Rational Unified Process is also available within IBM Rational Application Developer. View the Process Advisor.

__ a. Select **Help > Process Advisor**.

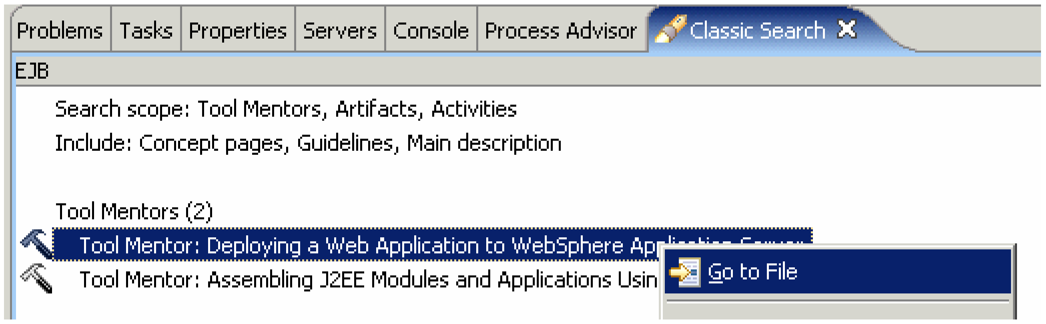
__ b. The **Process Advisor** view will open and different topics are displayed.

__ c. Click on the **Process Search** button which is the left-most button in the Process Advisor view.



__ d. A value can be searched for and a specific scope can be set. Enter EJB into the Search query field and click **Search**. The results will be displayed in the **Classic Search** view.

__ e. Right click on the first link under the **Tool Mentors**, then select **Go to File**. The topic will be displayed within the Process Browser.



EJB

Search scope: Tool Mentors, Artifacts, Activities

Include: Concept pages, Guidelines, Main description

Tool Mentors (2)

Tool Mentor: Deploying a Web Application to WebSphere Ap

Tool Mentor: Assembling J2EE Modules and Applications Usin

__ f. Click the Process Search view icon again. Select the **dW Search** tab. The **dW Search** tab allows you to search resources on the web which are hosted at **developerWorks**.

__ g. If you are connected to the Internet, enter **EJB** in the Search for field and click the **Search** button.

__ h. The results will be displayed again in the Classic Search view. Right click on the first result in the view and select **Go to File**.

__ i. A browser will be opened with the document displayed.

Part 3: Building the WebSphereBank Enterprise Application Project

____ 1. Create the appropriate Enterprise JavaBean and EAR projects for the application.

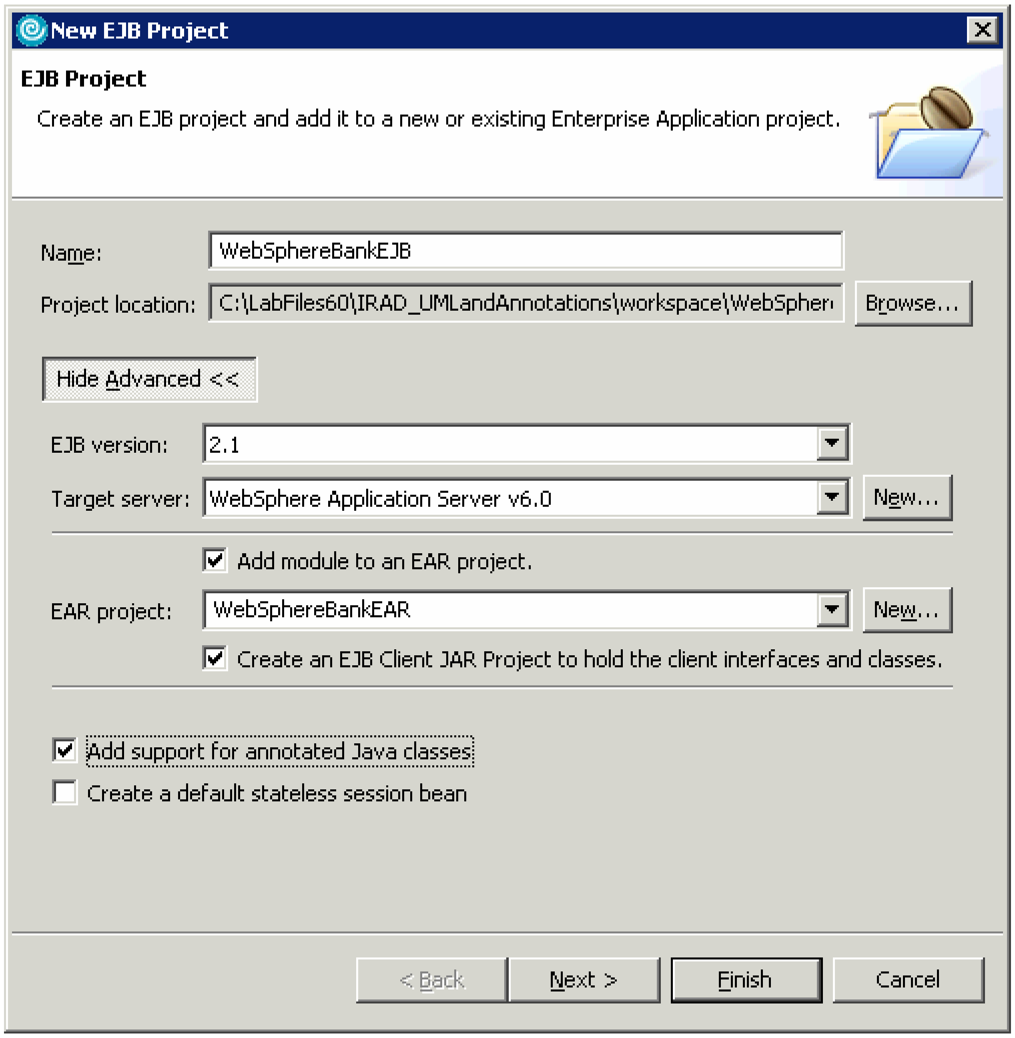
__ a. From the menu select **File > New > EJB Project**.

__ b. For the Name, enter **WebSphereBankEJB**.

__ c. Click **Show Advanced**.

__ d. Change the EAR project to **WebSphereBankEAR**.

__ e. Check the box **Add support for annotated Java classes**. This will setup Rational Application Developer to support annotation-based programming in the WebSphereBankEJB project. If the annotation support is not added at this time, when an enterprise bean is created, it can be added then.


New EJB Project

EJB Project

Create an EJB project and add it to a new or existing Enterprise Application project.

Name: WebSphereBankEJB

Project location: C:\LabFiles60\IRAD_UMLandAnnotations\workspace\WebSpher

EJB version: 2.1

Target server: WebSphere Application Server v6.0

[✓] Add module to an EAR project.

EAR project: WebSphereBankEAR

[✓] Create an EJB Client JAR Project to hold the client interfaces and classes.

[✓] Add support for annotated Java classes

[ ] Create a default stateless session bean

__ f. Click **Finish**.

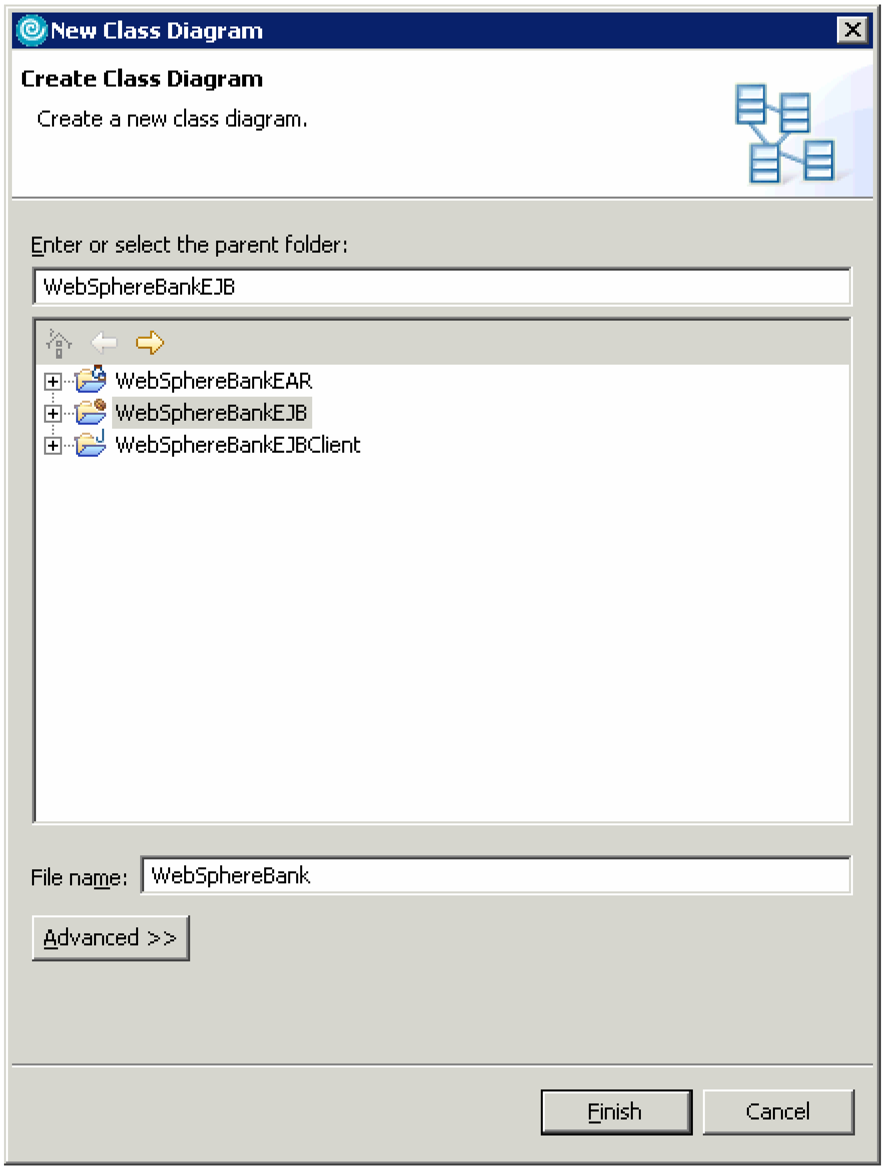
Part 4: Adding Enterprise Java Beans to WebSphereBank

____ 1. The main business object components of the application can be created using the UML editor. Create a UML class diagram editor.

__ a. In the Project Explorer open **EJB Projects** and right click on **WebSphereBankEJB** .

__ b. Select **New > Class Diagram** from the menu.

__ c. For the File name, enter **WebSphereBank**.



__ d. Click **Finish**. The UML editor for the project will be displayed.

____ 2. Create the Account entity bean.

__ a. From the Palette view, click **CMP 2.x Entity Bean** and click in the UML editor.

__ b. The Create an Enterprise Bean wizard will be started. For the Bean name, enter **Account**.

__ c. For the Default package, enter **com.ibm.websphere.samples.bank.ejb**.

__ d. Click the box for **Generate an annotated bean class**. This option will create the bean class with the appropriate annotation tags.

__ e. Click **Next**.

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

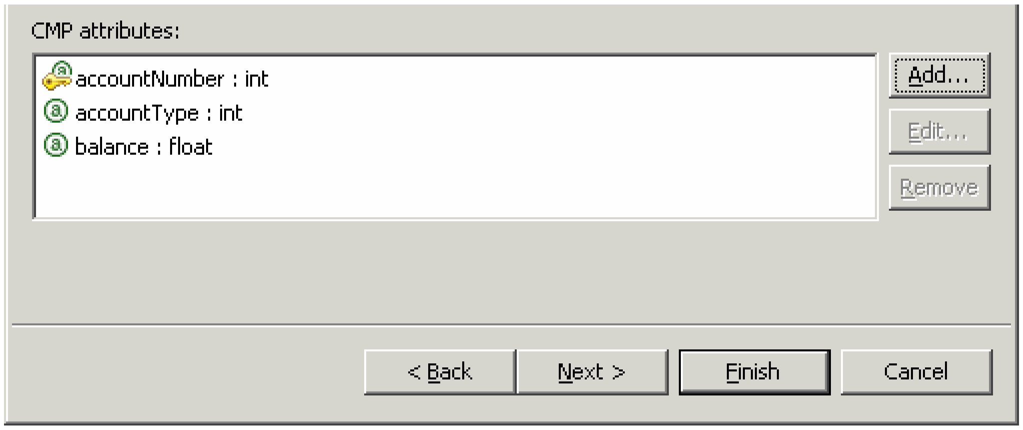
__ g. Select the **id:java.lang.Integer** entry in the list of CMP attributes and click **Remove**.

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

__ i. 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**.


CMP attributes:

accountNumber : int
accountType : int
balance : float

__ j. Click **Finish**. The Account bean will be displayed in the UML editor and created in the Enterprise JavaBean project.

____ 3. Create the Customer entity bean.

__ a. From the Palette, click **CMP 2.x Entity Bean** again and click in the UML editor.

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

__ c. Insure **com.ibm.websphere.samples.bank.ejb** is set for the Default package field and click **Next**.

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

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

__ f. Click the **Add** button next to the CMP attributes list.

__ g. 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**.

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

____ 4. Create the Transfer Stateless Session Bean.

__ a. From the Palette, click **Session Bean** and click in the UML editor.

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

__ c. Insure **com.ibm.websphere.samples.bank.ejb** is set for the Default package field.

__ d. Click **Next**.

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

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

__ g. Save the UML editor.

____ 5. Open the CustomerBean.java file.

__ a. From Project Explorer view, expand **EJB Projects > WebSphereBankEJB > ejbModule > com.ibm.websphere.samples.bank.ejb**.

__ b. Right click on CustomerBean.java and select **Open** from the menu.

____ 6. When the Customer bean was created, the option to create an annotated bean class was selected. With this option selected, a number of artifacts for the bean are defined with annotation tags and generated by the development environment before deployment. Much of the information, which defines the bean in the deployment descriptor, is also specified with annotation tags. The development environment is also responsible for setting the correct values in the deployment descriptor from these tags in order for the bean to be packaged correctly for deployment into a runtime environment. With annotation tags, a developer can focus attention on a single file rather than multiple Java source files and deployment descriptors. Inspect the annotations in the CustomerBean.java file.

__ a. If you scroll to the top of the CustomerBean file, you will see a large comment section. In this section there a number of annotation tags with details about the CustomerBean in general. Annotation tags can be set at the class, field, or method scope levels. These tags are at the class scope level. The first tag, @ejb.bean, contains parameters which define the overall bean. The name of the bean, type of bean, and JNDI name are some of the values.

```
/**
Bean implementation class for Entity Bean: Customer
*
@ejb.bean
name="Customer"
type="CMP"
cmp-version="2.x"
schema="Customer"
jndi-name="ejb/com/ibm/websphere/samples/bank/ejb/CustomerHome"
local-jndi-name="ejb/com/ibm/websphere/samples/bank/ejb/CustomerHome"
view-type="both"
reentrant="true"
*
```

__ b. The next two tags, **@ejb.home** and **@ejb.interface**, define the name of the files which will be the local home, local, home, and remote interfaces.

```
 * @ejb.home
 *    remote-class="com.ibm.websphere.samples.bank.ejb.CustomerHome"
 *    local-
class="com.ibm.websphere.samples.bank.ejb.CustomerLocalHome"
 *
 * @ejb.interface
 *    remote-class="com.ibm.websphere.samples.bank.ejb.Customer"
 *    local-class="com.ibm.websphere.samples.bank.ejb.CustomerLocal"
 *
```

__ c. The last tag, **@ejb.pk**, defines the class which represents the primary key for the CustomerBean.

```
 * @ejb.pk
 *  class="com.ibm.websphere.samples.bank.ejb.CustomerKey"
 *
```

__ d. If you select the **ejbCreate** method from the Outline view or scroll down to the ejbCreate method, you will see a tag which defines the method to be promoted to the local home interface for the bean.

```
/**
 * ejbCreate
 * @ejb.create-method
 *  view-type="local"
 */
public com.ibm.websphere.samples.bank.ejb.CustomerKey
      ejbCreate(long customerNumber) throws CreateException {
```

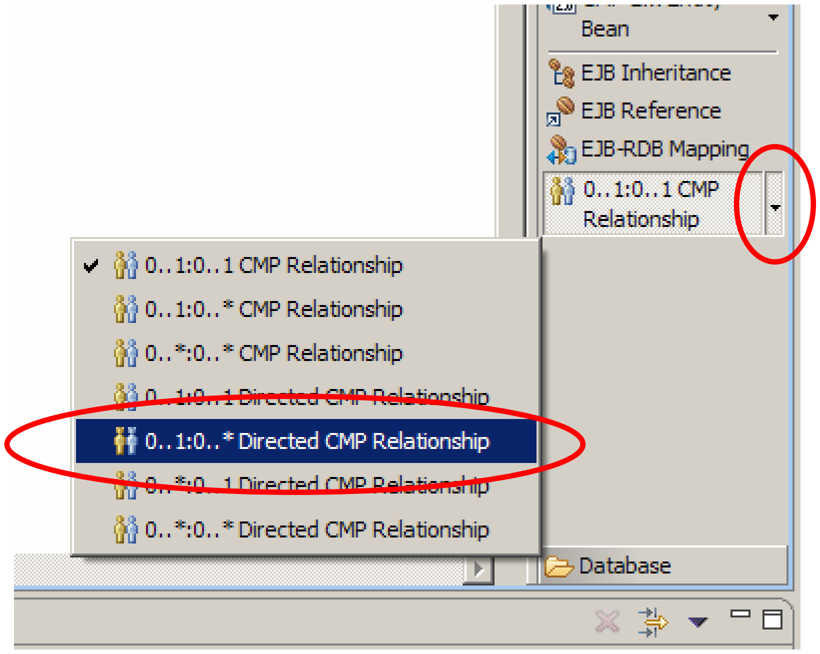
__ e. **Close** CustomerBean.java.

Part 5: Configure Container Managed Relationships

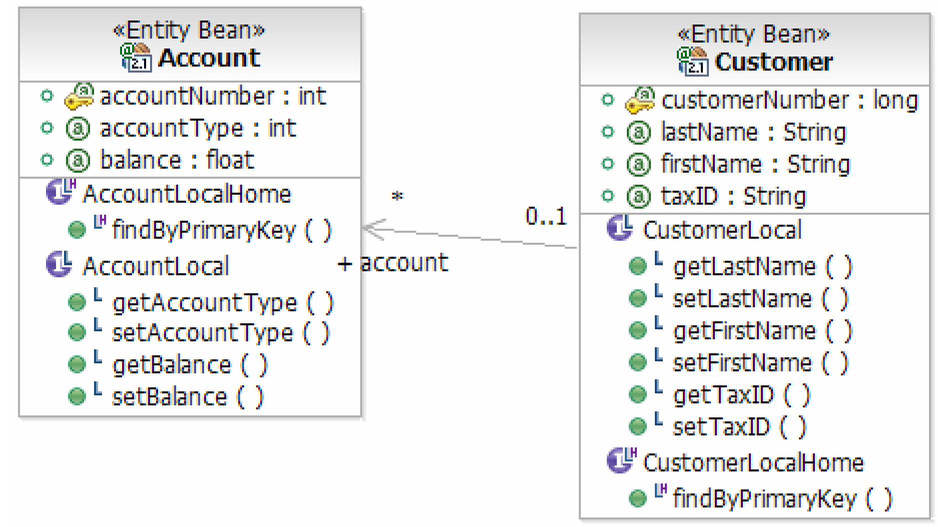
____ 1. Establish a relationship between the Customer and Account beans

__ a. Return to the UML Editor.

__ b. From the Palette, click on the drop down arrow next to the **0..1:0..1 CMP Relationship** option and select **0..1:0..* Directed CMP Relationship**.


Bean
EJB Inheritance
EJB Reference
EJB-RDB Mapping
0..1:0..1 CMP Relationship
0..1:0..1 CMP Relationship
0..1:0..* CMP Relationship
0..*:0..* CMP Relationship
0..1:0..1 Directed CMP Relationship
0..1:0..* Directed CMP Relationship
0..*:0..1 Directed CMP Relationship
0..*:0..* Directed CMP Relationship
Database

__ c. Click the Customer bean and hold as you drag to the Account bean before releasing. The relationship will be created.

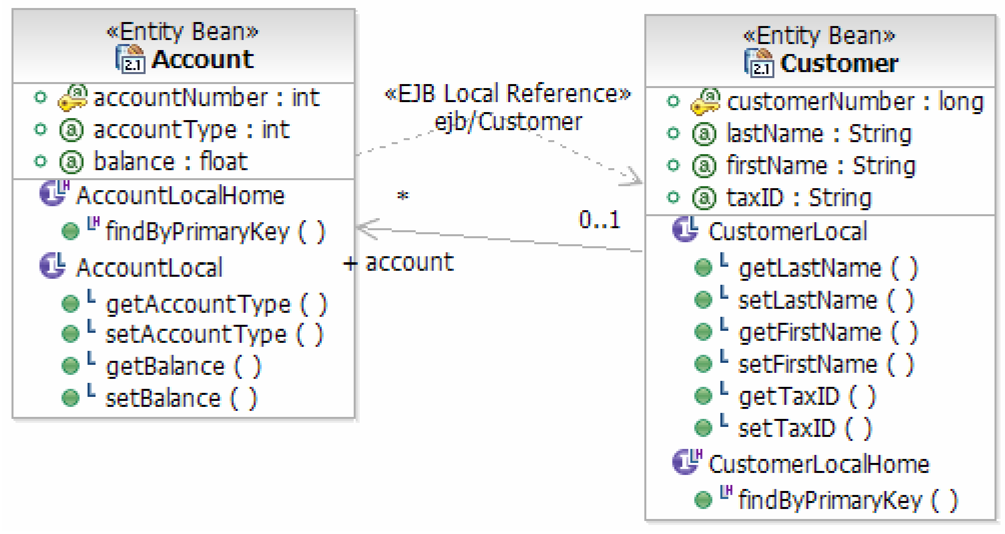


____ 2. Specify EJB References for the Account, Customer, and Transfer beans. These references are used for calling the different beans from the different interfaces. The following is a summary of the references that need to be configured.

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

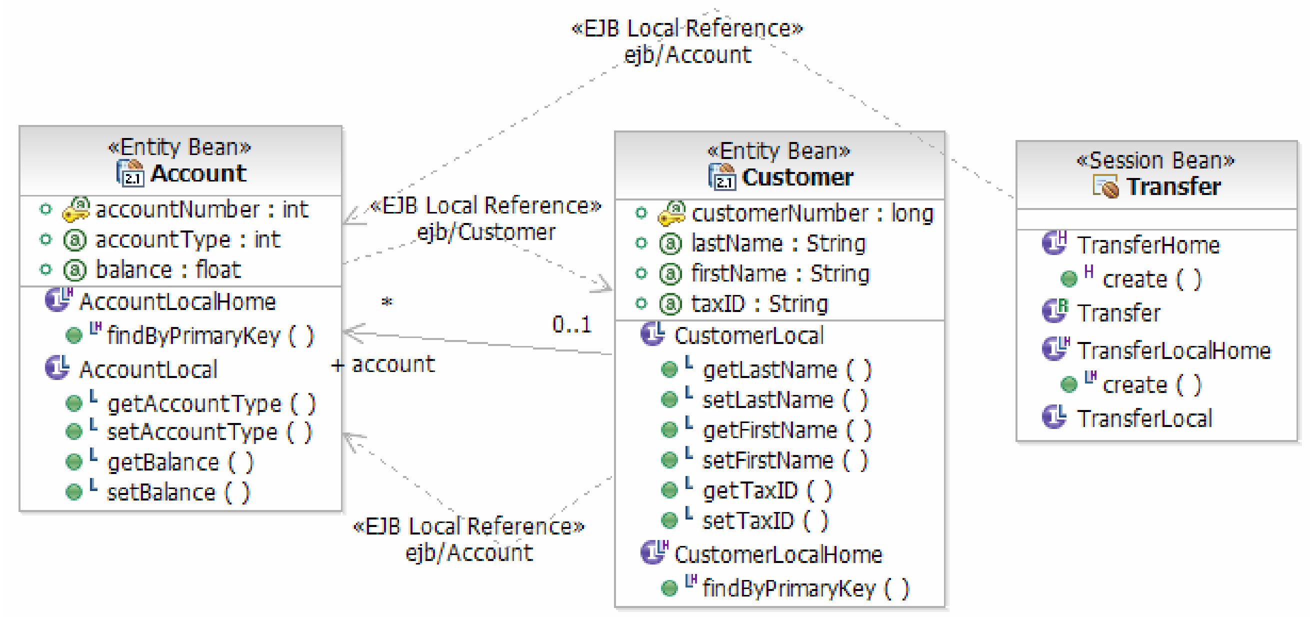
__ a. From the Palette view, click **EJB Reference**.

__ b. Click the Account bean and hold as you drag to the Customer bean before releasing.



__ c. When releasing, select **Create New EJB local reference**.

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

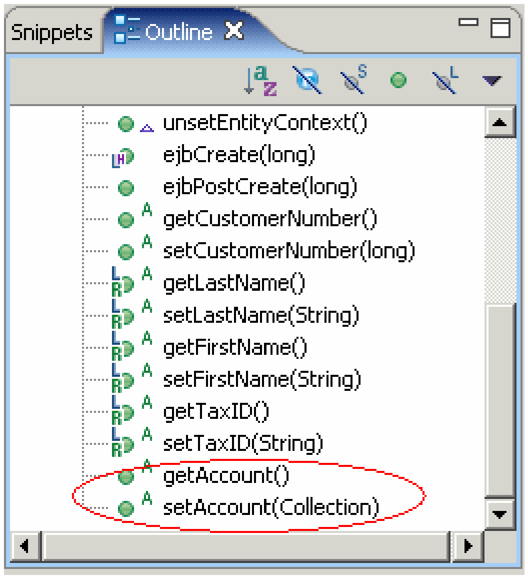


__ e. Save your progress.

_____ 3. Verify that the appropriate methods which support the container-managed relationship have been added to the CustomerBean.

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

__ b. From the Outline view verify that the **getAccount()** and **setAccount(Collection)** methods have been added to the class definition. These were added to support the container-managed relationship created in the UML editor.



unsetEntityContext()
ejbCreate(long)
ejbPostCreate(long)
getCustomerNumber()
setCustomerNumber(long)
getLastName()
setLastName(String)
getFirstName()
setFirstName(String)
getTaxID()
setTaxID(String)
getAccount()
setAccount(Collection)

Part 6: Adding Business Logic to our Enterprise Java Beans

____ 1. There are several classes that are referenced from within the business logic you are adding to your 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_UMLandAnnotations\BankUtilities.jar** and click **Open**.

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

__ e. You need to also import the BankUtilities into the WebSphereBankEJBClient project. From Project Explorer view, expand **Other Projects > WebSphereBankEJBClient** and right click on the **ejbModule** folder. Select **Import** from the context menu.

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

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

__ h. Verify that you are importing into the **WebSphereBankEJBClient/ejbModule** folder, and click **Finish**.

__ i. 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 you will add. First, there are two basic methods available for adding or subtracting funds from an account.

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

__ b. At the top of the class, add **TimedObject** to the class definition. This will allow for a timer to be associated with the Account bean.

```
public abstract class AccountBean implements EntityBean, TimedObject
{
```

__ c. Add the **add(), subtract()** and **Timer** methods to the AccountBean class. The code for this is included in **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet1.txt**. Copy all of the code included in this file into the AccountBean class under the line `private javax.ejb.EntityContext myEntityCtx;`.

__ d. The first statements you added are declarations for objects used in the business logic. The bundle is an object with messages available for the application and the CUSTOMER_JNDI_NAME is used for lookups of the CustomerBean.

```
private ListResourceBundle bundle = (ListResourceBundle)
ResourceBundle.getBundle("com.ibm.websphere.samples.bank.ejb.AccountR
esourceBundle");
private final static String CUSTOMER_JNDI_NAME =
"java:comp/env/ejb/Customer";
```

__ e. 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();
         }
```

__ f. 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();
        }
    }
```

__ g. 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_UMLandAnnotations\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(); // get this object's local
interface
            InitialContext initCtx = new InitialContext();
            CustomerLocalHome customerHome = (CustomerLocalHome)
initCtx.lookup(CUSTOMER_JNDI_NAME);

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

__ h. 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** or press **Ctrl+Shift+O**.

NOTE: You may be prompted to choose the appropriate Timer class. You should choose javax.ejb.Timer.

____ 3. Since you added a number of methods to the bean, you will need to promote these methods to the correct interface (Home, Local Home, Remote, or Local). With annotation-based programming, the promotion can be performed easily by adding in the correct tags to the methods in the bean Java source file.

__ a. In the Outline view, select **ejbCreate(int, int, float, long)**.

__ b. In the Javadoc comment above the method, add the tag **@ejb.create-method** below the method name. You can use Content Assist to help select the correct tag. Press **Ctrl+Space Bar** after typing the @ and select the tag from the list.

__ c. On the line below the @ejb.method tag, set the parameter **viewtype** to **"local"**. This will promote this create method to the local home interface.

```
/**
* ejbCreate
* @ejb.create-method
* view-type="local"
*/
public com.ibm.websphere.samples.bank.ejb.AccountKey ejbCreate(int
accountNumber, int type, float balance, long customerNumber) throws
javax.ejb.CreateException {
```

__ d. The following table summarizes the other methods you added to AccountBean, and indicates which methods should be promoted to the Local and/or Remote interfaces. In the Javadoc above each of the different methods, add the correct annotation to promote the method. (Remember to use Content Assist).

Promote local only

```
* @ejb.interface-method
* view-type="local"
```

Promote remote only

```
* @ejb.interface-method
* view-type="remote"
```

Promote to both local and remote

```
* @ejb.interface-method
* view-type="both"
```

Method	Local	Remote	view-type
add	YES	YES	"both"
subtract	YES	YES	"both"
myCreateTimer	YES	NO	"local"
myCancelTimer	YES	NO	"local"
getAccountNumber	YES	NO	"local"

__ e. **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.

____ 4. 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_UMLandAnnotations\snippets\snippet3.txt** file to the top of the TransferBean class as indicated below by the arrow.

```
public class TransferBean implements SessionBean {
private SessionContext mySessionCtx;          ←

private AccountLocalHome accountHome = null;
private final static String ACCOUNT_JNDI_NAME =
"java:comp/env/ejb/Account";

public float getBalance(int acctId) throws FinderException,
EJBException {
    AccountKey key = new AccountKey(acctId);
    AccountLocal fromAccount;
    try {
        fromAccount = accountHome.findByPrimaryKey(key);
        return fromAccount.getBalance();
    } catch (ObjectNotFoundException ex) {
        throw new FinderException("Account " + acctId + " does not
exist.");
    } catch (FinderException ex) {
        throw new FinderException("Account " + acctId + " does not
exist.");
    } catch (Exception r) {
        throw new EJBException();
    }
}

public void transferFunds(int fromAcctId, int toAcctId, float amount)
throws EJBException, InsufficientFundsException, FinderException {

    AccountKey fromKey = new AccountKey(fromAcctId);
    AccountKey toKey = new AccountKey(toAcctId);
    AccountLocal fromAccount, toAccount;
    try {
        fromAccount = accountHome.findByPrimaryKey(fromKey);
    } catch (ObjectNotFoundException ex) {
        throw new FinderException("Account " + fromAcctId + " does
not exist.");
    } catch (FinderException ex) {
        throw new FinderException("Account " + fromAcctId + " does
not exist.");
    } catch (Exception r) {
        throw new EJBException();
    }

    try {
        toAccount = accountHome.findByPrimaryKey(toKey);
    } catch (ObjectNotFoundException ex) {
        throw new FinderException("Account " + toAcctId + " does not
exist.");
```

```
    } catch (FinderException ex) {
        throw new FinderException("Account " + toAcctId + " does not
exist.");
    } catch (Exception r) {
        throw new EJBException();
    }

    try {
        toAccount.add(amount);
        fromAccount.subtract(amount);
    } catch (InsufficientFundsException ex) {
        mySessionCtx.setRollbackOnly();
        throw new InsufficientFundsException("Insufficient fund in "
+ fromAcctId);
    } catch (Exception r) {
        throw new EJBException();
    }
}

private AccountLocalHome getAccountHome() throws RemoteException {
    try {
        InitialContext initCtx = new InitialContext();
        Object objref = initCtx.lookup(ACCOUNT_JNDI_NAME);

        return(AccountLocalHome) objref;
    } catch (NamingException ne) {
        ne.printStackTrace();
        throw new RemoteException("Error looking up AccountHome
object: " + ne.getMessage());
    }
}
```

__ 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_UMLandAnnotations\snippets\snippet4.txt** file. Copy this code in between the open and close brackets of the ejbCreate method.

```
public void ejbCreate() throws CreateException {

    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** or **Ctrl+Shift+O**.

NOTE: You may be prompted to choose the appropriate ObjectNotFoundException class. You should choose javax.ejb.ObjectNotFoundException.

__ e. 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 specify the correct annotation tags. Locate and select the method in the Outline view and add the appropriate tag above the method in the editor.

Promote to both local and remote

```
* @ejb.interface-method
* view-type="both"
```

Method	Local	Remote	view-type
getBalance	YES	YES	"both"
transferFunds	YES	YES	"both"

__ f. Save and **Close** the **TransferBean.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.

____ 5. 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 new **ejbCreate** and **ejbPostCreate** methods.

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

__ b. You will find the code that you are to add to the CustomerBean class in the **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet5.txt** file. This file includes the 4 methods discussed above. Add this code to the top of the CustomerBean class, as indicated by the arrow below.

```
public abstract class CustomerBean implements EntityBean {
private javax.ejb.EntityContext myEntityCtx;

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

public Vector getOwnedAccountNumbers() { //returns all the acct
numbers to the client - for CMR verification
    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;
    }

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 {
}
```

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

NOTE: You may be prompted to choose the appropriate Iterator class. You should choose **java.util.Iterator**. Also note there may still be some errors indicated in the CustomerBean class, but these errors will be addressed in the following steps.

__ d. Since you added the **ejbCreate(CustomerKey, String, String, String)** method, you will need to promote this method to the Local Home interface. In the **Outline view**, select **ejbCreate(CustomerKey, String, String, String)**. Add the appropriate annotation tag above the method to promote the method.

```
/**
  * ejbCreate
  * @ejb.create-method
  * view-type="local"
  */
 public com.ibm.websphere.samples.bank.ejb.CustomerKey
ejbCreate(CustomerKey customerKey, String name, String lname, String
taxID) throws javax.ejb.CreateException {
```

__ e. The following table summarizes the methods you added in Step 5a, and indicates which methods should be promoted to the Local and/or Remote interfaces. To promote these methods to the Local and Remote interface specify the correct annotation tags. Locate and select the method in the Outline view and add the appropriate tag above the method in the editor.

Promote local only

```
* @ejb.interface-method
* view-type="local"
```

Promote remote only

```
* @ejb.interface-method
* view-type="remote"
```

Method	**Local**	**Remote**	**view-type**
addAccount	YES	NO	“local”
getOwnedAccountNumbers	NO	YES	“remote”
getAccountsList	NO	YES	“remote”

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

Part 7: Adding EJB QL Statements

EJB QL can be used to search for CMP entity beans and to search across relationships. Using annotations, find and select methods can be easily defined.

____ 1. Create a find method for locating the appropriate Customer when with a specified last name.

__ a. Open **CustomerBean.java**.

__ b. At the top of the class, tags can be entered for defining finder methods. At the end of the comment section, add the following tag and parameters using Content Assist to define a query which will return a collection of customers with a particular lastname passed as a parameter. The query statement can also be copied from the file **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet6.txt**.

```
*
* @ejb.finder
* signature="java.util.Collection findByLastName(java.lang.String
name)"
* query="SELECT OBJECT (c) FROM Customer c WHERE c.lastName LIKE ?1
ORDER BY c.lastName"
```

__ c. The signature parameter specifies the actual signature for the method which is added to the home and local home interfaces of the bean. The query parameter contains the query which is placed in the ejb-jar.xml deployment descriptor.

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

__ a. Scroll to the bottom of the class.

__ b. Add the following method declaration to the class.

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

__ c. Above the method, add the following tag and parameters using Content Assist. The query statement text area can be copied from the file **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet6.txt**.

```
/**
* @ejb.select
*   query="SELECT a.accountNumber FROM Customer c, IN(c.account) a
WHERE c.customerNumber = ?1 AND a.balance > ?2"
*/
```

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

__ a. Beneath the method you just added add another method declaration.

```
public abstract int ejbSelectAccounts(long customerNumber) throws
javax.ejb.FinderException;
```

__ b. Above the method, add the following tag and parameters using Content Assist. The query statement text area can be copied from the file **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet6.txt**

```
/**
* @ejb.select
*   query="SELECT COUNT(a) FROM Customer c, IN(c.account) a WHERE
c.customerNumber = ?1"
*/
```

____ 4. Because select methods cannot be directly exposed to the client, you will need to add two methods that expose the ejbSelect methods. In this simple example, you need business logic methods that expose the results so a web module can use them.

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

__ b. Cut and paste the **getAccountsByBalance** and the **getAccountsCount** methods provided in **<LAB_FILES>\IRAD_UMLandAnnotations\snippets\snippet6.txt**, at the bottom of the CustomerBean class in the Java editor.

```
public Vector getAccountsByBalance(float minimumBalance) {
final String methodName = "getAccountByBalance";

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() {
final String methodName = "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 Remote interfaces. To promote these methods to the Local and Remote interface specify the correct annotation tags. Locate and select the method in the Outline view and add the appropriate tag above the method in the editor.

Promote remote only

```
* @ejb.interface-method
* view-type="remote"
```

Promote to both local and remote

```
* @ejb.interface-method
* view-type="both"
```

Promote both

Method	Local	Remote	view-type
getAccountsByBalance	NO	YES	“remote”
getAccountsCount	YES	YES	“both”

____ 5. **Save** and **Close** the CustomerBean.java file.

____ 6. Review the deployment descriptor code.

__ a. In the Project Navigator view, expand **EJB Projects > WebSphereBankEJB**.

__ b. Open the **Deployment Descriptor**.

__ c. On the Overview tab, select **Customer** under the Enterprise JavaBeans section.

__ d. The Bean tab should be displayed and the Customer bean selected. On the upper right section, scroll down to the list of Queries. Notice the find and two select methods you created have been registered in the deployment descriptor.

__ e. Close the EJB Deployment Descriptor editor.

Part 8: Complete Application

____ 1. Remove the Cloudscape mappings generated when the entity beans were created. These mappings are incorrect and can only be recreated after removing the generated mappings.

__ a. Expand **EJB Projects > WebSphereBankEJB > ejbModule > META-INF > backends**.

__ b. Delete the **CLOUDSCAPE_V51_1** folder. This will delete the mappings.

____ 2. Verify that there are no errors listed in the **Problems** view. You should only see informational messages listed.

____ 3. Specify the JNDI name for the CMP connection factory binding used by the entity beans for database persistence.

__ a. Open the **EJB Deployment Descriptor**.

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

__ c. Scroll to the bottom and 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**

__ d. **Save** and **Close** the EJB deployment descriptor.

____ 4. Import the WebSphereBankWeb 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_UMLandAnnotations\WebSphereBankWeb.war**. Click **Open**.

__ d. The Web Project should be set to **WebSphereBankWeb**, and select the EAR project: **WebSphereBankEAR** 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, right-click on one of the errors and select **Quick Fix**.

__ g. A box of possible solutions will be displayed. Select **Add project 'WebSphereBankEJBClient' to build path of 'WebSphereBankWeb'**. Click **OK**.

____ 5. 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_UMLandAnnotations\FindAccounts.jar** and click **Open**.

__ d. The Application Client project should be set to **FindAccounts**, and the EAR project should be **WebSphereBankEAR**. 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, right-click on one of the errors and select **Quick Fix**.

__ g. A box of possible solutions will be displayed. Select **Add project 'WebSphereBankEJBClient' to build path of 'WebSphereBankWeb'**. Click **OK**.

__ h. Repeat Steps 5a-g above for the **GetAccounts** application client module. For this module you will import the file: **<LAB_FILES>\IRAD_UMLandAnnotations\GetAccounts.jar**.

____ 6. Export WebSphereBank EAR file.

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

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

__ c. Browse to **<LAB_FILES>\IRAD_UMLandAnnotations** 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 banking application which highlights several important concepts associated with the EJB specification. Using rapid application development techniques such as UML and annotation based programming, IBM Rational Application Developer v6, eases the development effort. By providing a visual development and a simplified format with annotations, developers can quickly and easily build J2EE applications.

Solution Instructions

____ 7. Import the WebSphereBank application into Rational Application Developer for testing.

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

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

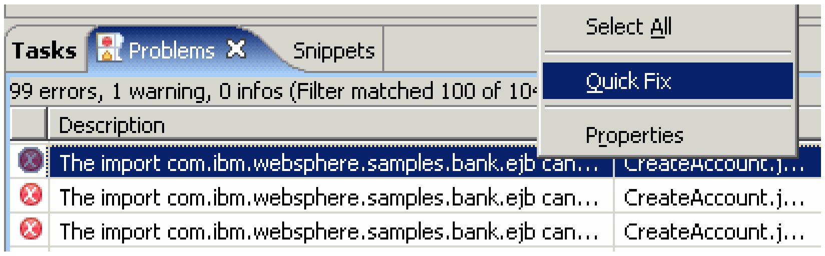
__ c. Select **Browse...** and navigate to **<LAB_FILES>\IRAD_UMLandAnnotations\solution\WebSphereBank.ear** and select **Open**.

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

__ e. Click **Finish**.

____ 8. **Add Java Build Path** to clean up errors. If you select the Problems tab, you will notice a list of errors and warnings.

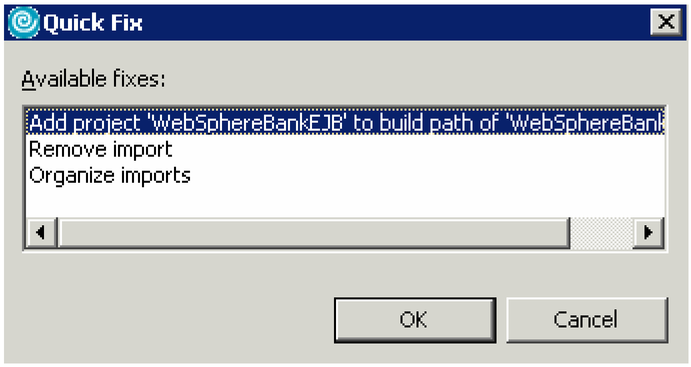
__ a. Right click on an error and select **Quick Fix.**



99 errors, 1 warning, 0 infos (Filter matched 100 of 10

The import com.ibm.websphere.samples.bank.ejb can... CreateAccount.j...
The import com.ibm.websphere.samples.bank.ejb can... CreateAccount.j...
The import com.ibm.websphere.samples.bank.ejb can... CreateAccount.j...

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



Add project 'WebSphereBankEJB' to build path of 'WebSphereBank
Remove import
Organize imports

____ 9. Explore the WebSphereBankEJB project, and the various EJBs developed in this lab.

Trademarks and Disclaimers