Exercise 2:

Creating a Hello World Stateless Session EJB™

Case Study: Creating a Hello World Stateless Session EJB

Estimated completion time: 1 hour 15 minutes.

In this lab, you create a stateless session EJB™ that prints out the phrase “Look, Mom, I'm an EJB!”. The purpose of this lab is to introduce you to session EJBs and provide you with a simple application to deploy to the J2EE runtime environment. As shown in Figure 1, the `HelloWorld` EJB runs inside the J2EE environment and is accessed from a test client running from within the IDE. The `HelloWorld` client invokes the `HelloWorld` session EJB’s `sayHello` method, which returns a simple string message to the client bean.

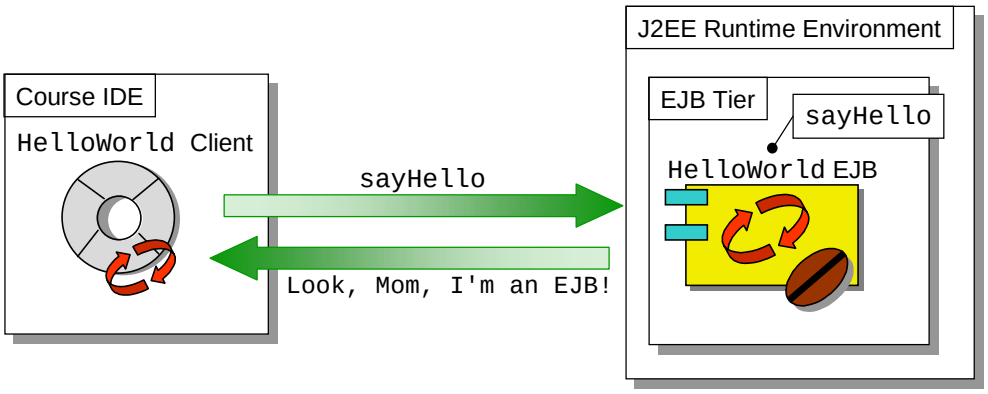

Figure 1 - Accessing the `HelloWorld` Session EJB

Using the instructions given below, you create the session bean and test client using the course IDE, assemble and deploy the session bean into a J2EE runtime environment using a J2EE assembly/deployment tool, and then test the bean.

There are three sections to this lab exercise. In the first section, you code the `HelloWorld` EJB’s home and remote interfaces and create a bean class. The bean class implements the bean’s business and life-cycle methods. In Section 2, you create a test client to access the `HelloWorld` session bean. Finally, in Section 3, you deploy the session bean and test your work.

After completing this lab, you will be able to:

- Create a stateless session EJB
- Use JNDI™ to access an EJB running in a J2EE™ container
- Create and deploy a J2EE application containing session EJBs
- Configure the IDE’s execution settings to access J2EE components from within the IDE

Discussion Topics:

- What additional functionality could be obtained by making the `HelloWorld` session bean stateful?
- What is JNDI? How is it used in this lab exercise?
- How must you configure the course IDE to access EJB components running in the J2EE Reference Implementation runtime environment?

Important Terms and Concepts

Refer to the JDK Javadoc or the lab Appendix for information on the important classes and methods used in this exercise, including parameters and return values.

javax.ejb

```
public class CreateException extends java.lang.Exception
public class EJBException extends java.lang.RuntimeException
public class RemoveException extends java.lang.Exception
public interface SessionBean extends EnterpriseBean
   void ejbActivate()
   void ejbPassivate()
   void ejbRemove()
   void setSessionContext(SessionContext ctx)
public interface SessionContext extends EJBContext
   EJBObject getEJBObject()
```

java.rmi

```
public class RemoteException extends IOException
```

javax.rmi

```
public class PortableRemoteObject extends java.lang.Object
   static java.lang.Object narrow(java.lang.Object narrowFrom,
                                  java.lang.Class narrowTo)
```

javax.naming

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

Setup Instructions

There is no special setup required for this lab exercise. Use the “Guidelines for Creating a Session EJB” or the accompanying course material as a resource when completing this exercise. You can view an example of the completed exercise in the “Solution Code” section of this lab module.

Section 1

Define the Session EJB's Home and Remote Interface

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

1. Create a new Java package in the IDE workspace named "Server".
2. Create a session home interface for the `HelloWorld` session EJB in the `Server` package.
   - Name: `HelloHome`
   - Creator methods:
     - `create`
3. Create a remote interface for the `HelloWorld` EJB in the `Server` package.
   - Name: `Hello`
   - Business methods:
     - `String sayHello` – Returns a message to the client bean

Define the Session EJB's Bean Class

4. In the `Server` package, create a skeleton for the `HelloWorld` EJB bean class containing the methods that implement the corresponding method stubs on the home and remote interfaces. Include stubs for any additional methods required by the container and implemented in the bean class. Do not code any of the bean class's method bodies at this time.
   - Name: `HelloEJB`

Implement the Bean Class Methods

5. Code the bean class's `sayHello` method to return the string "Look, Mom, I'm an EJB!".

Compile

6. Build the files contained in the `Server` package. The entire package should compile without errors. Fix any errors before continuing with this exercise. Verify your work against the solution code provided for this module if necessary.

Discussion Topics:

- Explain when you might add multiple creator methods to a session EJB. What does the existence of multiple creator methods imply about the nature of a session EJB?
- Discuss how the session bean life cycle impacts the operation of the `HelloWorld` session bean.
- Describe the process of connecting to a session EJB from within a client application.
- When must a bean developer implement the method bodies for `ejbPassivate`, `ejbActivate`, and `ejbRemove`?
- Explain the function of the `setSessionContext` method.

Section 2

Implement the Test Client

1. Create a new Java package in the IDE workspace named "Client".
2. Create a class inside the `Client` package using the criteria listed below:
   - Type: `Main`
   - Name: `HelloWorld`
   - Imports: `javax.naming.*`, `javax.rmi.*`, and `Server`
3. Code the test client's `main` method to retrieve a reference to the `HelloWorld` EJB's remote interface using the JNDI name "helloServer", create an EJB instance, invoke the EJB's `sayHello` method and display the EJB's output, and finally remove the EJB. Be sure to catch and handle any exceptions that might occur when making the remote connection with the session EJB.

Compile

4. Compile the `HelloWorld` test class. Correct any errors before continuing with this exercise.

Discussion Topics:

- What types of exceptions might a client component receive when invoking methods on a session EJB?
- Why must a client "remove" a session EJB after it is through using it?

Section 3

Deploy and Test

In this section, you deploy and test your session EJB. Consult the "Assembly and Deployment Guidelines" for information on deploying EJBs into the J2EE container used for this course. Use the "IDE Configuration Guidelines" to assist you in modifying the IDE runtime configuration to enable access to components running in the J2EE container.

1. Using the J2EE assembly/deployment tool provided for this course, assemble and deploy the `HelloWorld` session EJB into the J2EE runtime environment. Use "helloServer" as the JNDI name for the session EJB. To assemble/deploy the session bean, you must:
    - Start the J2EE runtime, Cloudscape database, and deployment tool.
    - Use the deployment tool to:
        - Create a new application named "HelloApp". Store the application `.ear` file in the *<J2EEAppHome>* directory.
        - Add the `HelloWorld` session bean to the application:
            - **New Enterprise Bean Wizard - EJB JAR**
                - Enterprise Bean Will Go In: `HelloApp`
                - JAR Display Name: `HelloJAR`
                - Contents - Add:
                    - Choose the system workspace as the root directory; for example, `C:\`*<IDEHome>*`\Development`
                    - Open the `Server` package.
                    - Select and add `Hello.class`, `HelloEJB.class`, and `HelloHome.class` from the `Server` package.
            - **New Enterprise Bean Wizard - General**
                - Enterprise Bean Class: `Server.HelloEJB`
                - Home Interface: `Server.HelloHome`
                - Remote Interface: `Server.Hello`
                - Display Name: `HelloBean`
                - Bean Type: Session/Stateless
            - **New Enterprise Bean Wizard - Transaction Management**
                - Container-Managed Transactions: Checked
                - `sayHello` Transaction Type: Required
        - Deploy the application:
            - Select the option to Return Client Jar, and store the client `.jar` file in the *<J2EEAppHome>* directory.
            - Use "helloServer" as the JNDI name for the `HelloWorld` session bean.
2. Configure the test client with the path to the client `.jar` file, and execute the test client. View the results in the IDE's run console.

Solution Code: Section 1

```
// HelloHome.java

package Server;

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

public interface HelloHome extends EJBHome {
   Hello create() throws RemoteException, CreateException;
}

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

// hello.java

package Server;

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

public interface Hello extends EJBObject {
   public String sayHello() throws RemoteException;
}

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

// HelloEJB.java

package Server;

import javax.ejb.*;

public class HelloEJB implements SessionBean {

   public String sayHello() {
      return "Look, Mom, I'm an EJB!";
   }

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

Solution Code: Section 2

// HelloWorld.java

```
package Client;

import javax.naming.*;
import javax.rmi.*;
import Server.*;

public class HelloWorld extends Object {

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

   /**
   * @param args the command line arguments
   */
   public static void main (String args[]) {
      try {
         //get home object
         Context c = new InitialContext();
         Object result = c.lookup("helloServer");
         // cast the object to a home object
         HelloHome h =
            (HelloHome)javax.rmi.PortableRemoteObject.narrow
                (result,HelloHome.class);
         //get session object and make request
         Hello r = h.create();
         System.out.println("Server says:" + r.sayHello());
         // remove session bean instance
         r.remove();
      }
      catch (Exception e) { System.out.println("Error: " + e); }
   }
}
```

Answers for Discussion Topics:

- What additional functionality could be obtained by making the HelloWorld session bean stateful?

  Answer: Because the functionality of the HelloWorld session bean is somewhat trivial, the additional benefit derived from making the bean stateful is somewhat limited. Creating a stateful HelloWorld session bean would allow you to pass information to the bean about the client; for example, the user's name that could possibly be incorporated in the returned message.

- What is JNDI? How is it used in this lab exercise?

  Answer: JNDI stands for Java™ Naming and Directory Interface. JNDI defines a directory service and naming facility used by Java applications that is independent of any specific directory service implementation. Using JNDI, programmers can have seamless access to directory objects through multiple naming facilities. In this lab exercise, the application client uses JNDI to access the HelloWorld EJB running in the J2EE runtime environment.

- How must you configure the course IDE to access EJB components running in the J2EE Reference Implementation runtime environment?

  Answer: The IDE must be configured with the path to the J2EE runtime and client application JAR files. See the "IDE Guidelines" for more information about this topic.

- Explain when you might add multiple creator methods to a session EJB. What does the existence of multiple creator methods imply about the nature of a session EJB?

  Answer: Statefull session bean fields are initialized using the creator method parameter values. Having multiple creator methods implies that a session bean is stateful. Stateless session beans cannot have parameterized creator methods. Passing a parameter to a session bean in the creator method implies that the bean state might vary based on how it is invoked.

- Discuss how the session bean life cycle impacts the operation of the HelloWorld session bean.

  Answer: No direct impact. The container manages the life cycle. The EJB contains no data elements requiring special treatment when passivating or activating the EJB.

- Describe the process of connecting to a session EJB from within a client application.

  Answer: Look up the home interface, call the EJB's `create` method, execute one or more business methods, call the EJB's `remove` method when through using the EJB.

- When must a bean developer implement the method bodies for `ejbPassivate`, `ejbActivate`, and `ejbRemove`?

  Answer: The method, `ejbPassivate`, is called by the container prior to passivating a session bean instance. It is used by the bean developer to perform special actions required to place all bean fields in a serializable state and release any acquired resources prior to passivation. The method `ejbActivate` is called by the container when activating a passivated bean instance and is used to restore non-serializable bean fields and acquired resources. All session beans must implement these methods even if they contain no code in the method body. A container does not activate or passivate stateless session beans. The method `ejbRemove` is called by the container and should free any resources held by the bean class. The bean provider must implement `ejbRemove` even if the method body contains no code.

- Explain the function of the `setSessionContext` method.

  Answer: The `setSessionContext` method, called by the EJB container after instantiating the EJB object and bean class instance, is used to initialize the bean instance. It is common practice to place code in this method that stores the `SessionContext` to an EJB instance variable. The `SessionContext` provides access to the runtime session context that the container provides for a session enterprise Bean instance.

- What types of exceptions might a client component receive when invoking methods on a session EJB?

  Answer: The client should catch and handle the `RemoteException` and any application exceptions thrown by the session EJB.

- Why must a client "remove" a session EJB after it is through using it?

  Answer: Removing a session EJB allows the container to return the bean to the pool, making it available for use by other clients.

Filename:	lab02.doc
Directory:	H:\FJ310_revC_0301\BOOK\EXERCISE\Exercise_Originals
Template:	C:\Program Files\Microsoft Office\Templates\Normal.dot
Title:	Case Study: The New User Interface
Subject:	
Author:	jdibble
Keywords:	
Comments:	
Creation Date:	04/24/01 10:04 AM
Change Number:	3
Last Saved On:	04/24/01 12:32 PM
Last Saved By:	Nancy Clarke
Total Editing Time:	2 Minutes
Last Printed On:	04/24/01 12:36 PM

As of Last Complete Printing

Number of Pages:	10
Number of Words:	1,962 (approx.)
Number of Characters:	11,185 (approx.)