Appendix K: Build a Java EE 6 Database Application with SQL Server Database

CONTENT

K.1 Creating a Java EE 6 Web Application Project …….2
K.2 Creating the Entity Classes from the Database ……..3
K.3 Creating the Enterprise Java Beans ……4
K.4 Using JavaServer Faces (JSF) 2.0 ……..8
K.5 Creating the Faculty Managed Bean ……..9
K.6 Creating the Faculty Listing Web Page ……11
K.7 Add the JDBC Driver for the SQL Server Database into the Project ……13
K.8 Building and Running the First Java EE 6 Web Page ……...14
K.9 Creating the Faculty Details Web Page ……15
K.10 Creating and Editing the faces-config.xml Configuration File …….…….17
K.11 Editing the General Web Application Configuration File web.xml ……..……22
K.12 Modifying the FacultyList and FacultyDetails Pages to Perform Page Switching ……23
K.13 Building and Running the Entire Java EE 6 Project ……..24

K.1 Creating a Java EE 6 Web Application Project

Perform the following operations to create a new Java EE 6 Web application project named `JavaEEWebDBFaculty`:

1. Launch the NetBeans 6.8 IDE.
2. Choose `File > New Project` (`Ctrl-Shift-N`) from the main menu.
3. Select `Enterprise Application` from the `Java EE` category and click on the `Next`.
4. Type `JavaEEDBFaculty` for the project name and set the desired project location.
5. Deselect the `Use Dedicated Folder` option, if selected. Click on the `Next`.
6. Set the server to `GlassFish v3` and set the Java EE Version to `Java EE 6`. Keep all other default settings and click on the `Finish` button.

Your finished `New Enterprise Application` window should match one that is shown in Figure K.1.

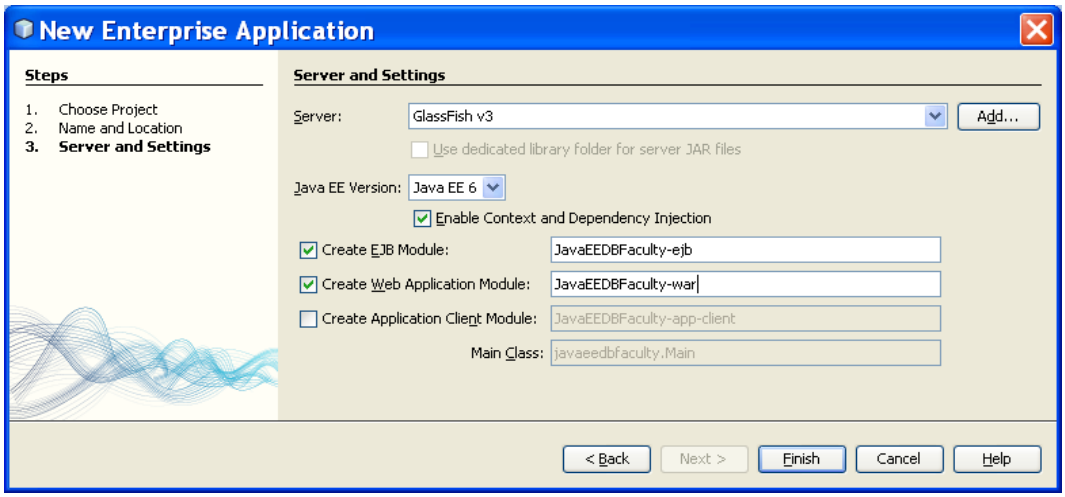

Figure K.1 The finished New Enterprise Application window.

NetBeans will create three projects namely `JavaEEDBFaculty` (Enterprise Application project), `JavaEEDBFaculty-ejb` (EJB project) and `JavaEEDBFaculty-war` (Web project), as shown in Figure K.2.

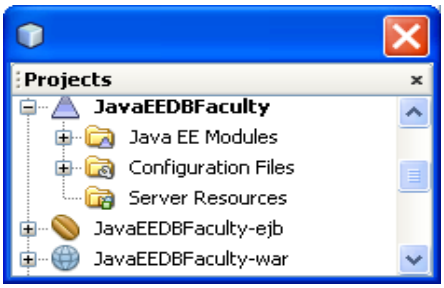

Figure K.2 Created three projects by NetBeans IDE.

Next let's create our entity classes to map our sample database and tables since the Session Beans are responsible for manipulating the data and they will be created in the EJB project.

K.2 Creating the Entity Classes from the Database

Perform the following operations to create our entity classes for our sample database:

1. In the `Projects` window, right-click on the `JavaEEDBFaculty-ejb` project and select the `New > Entity Classes from Database...` item from the popup menu.
2. Check the `Data Source` Radio button, click on the dropdown arrow and select the `New Data Source` from the corresponding dropdown list.
3. On the opened `Create Data Source` dialog, enter `CSE_DEPT` into the JNDI Name field and click on the dropdown arrow on the `Database Connection` combo box, and select our sample database CSE_DEPT connection URL `jdbc:sqlserver://localhost\SQLEXPRESS:5000;databaseName=CSE_DEPT [ybai on dbo]`. Click on the `OK` button to close this dialog box.

Note: Make sure that you have installed the JDBC Driver for our sample SQL Server database CSE_DEPT we built in Chapter 2 and connected it with the NetBeans 6.8 IDE in the Services window. Refer to section 6.2.1.2 in Chapter 6 to complete these steps.

4. Under the `Available Tables` list box, select `Faculty` and click on `Add` button so that it appears in the `Selected Tables` list box. Your `New Entity Classes from Database` window should match one that is shown in Figure K.3. Click on the `Next` button to continue.

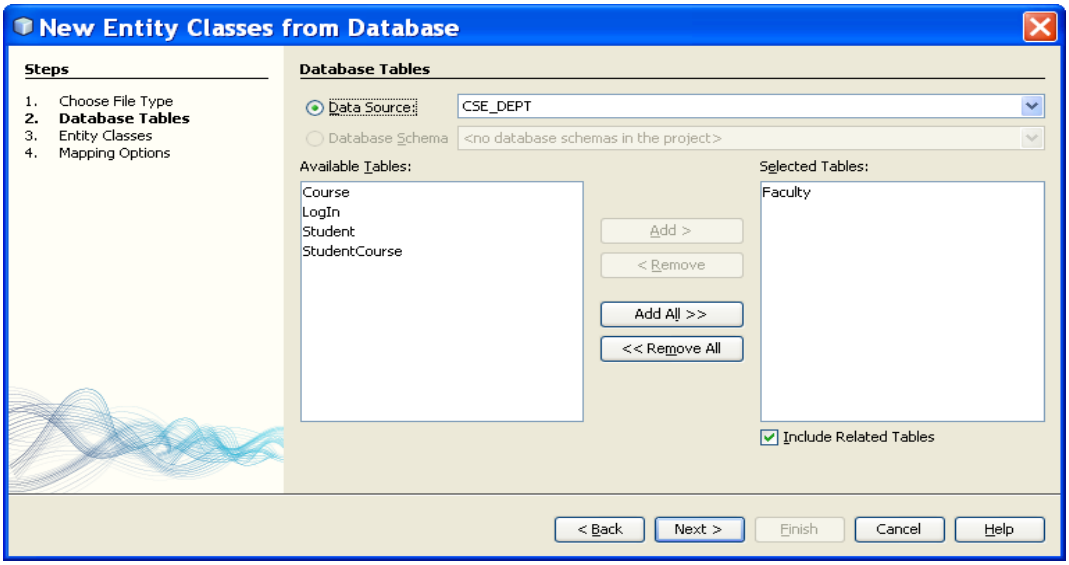

Figure K.3 The New Entity Classes from Database window.

5. Click on the `Create Persistence Unit` button and select `CSE_DEPT` as the `Data Source`. Leave the rest as default as shown in Figure K.4 and click on the `Create` button to continue.
6. Provide a package name, `com.javaeedbfaculty.entity` in the `Package` field and click on the `Next` button.
7. Change the `Collection Type` to `java.util.List` and click on the `Finish` button to complete this entity class creation process.

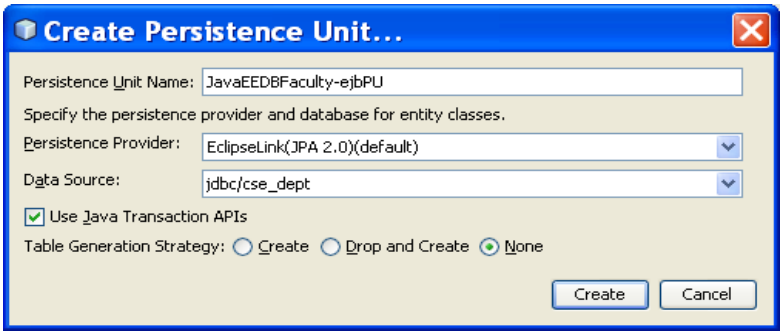

Figure K.4 The Create Persistence Unit dialog.

You can find that one entity class, `Faculty.java` has been created under the `Source Packages`, `com.javaeedbfaculty.entity`, in the `Projects` window, which is shown in Figure K.5.

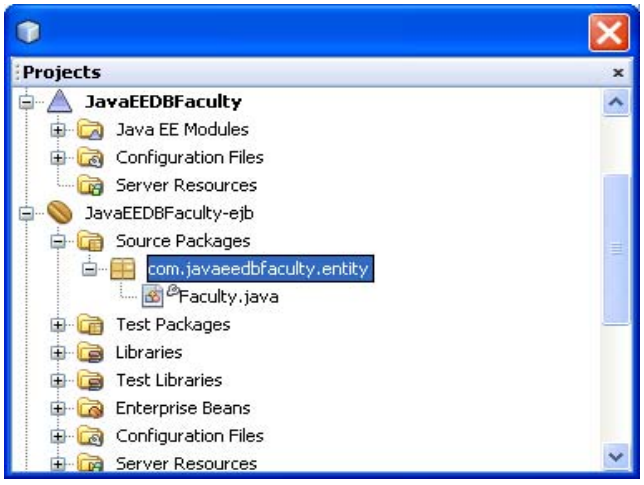

Figure K.5 The new created entity class Faculty.java.

Next let's create the Java Beans to perform communication functions between the JSF pages and Java Persistence API to make the data actions against our sample database.

K.3 Creating the Enterprise Java Beans

Now that we have the Entity classes, the next step is to create the Session (Stateless) Bean, `FacultySession` that will manipulate and provide the Retrieving functionality on the `Faculty`

object. In this application, the client that uses this function is the JSF pages. One of the benefits of doing this (i.e. to provide the functionalities in the EJB layer) is reusability because the same functions can be used by more than one JSF pages, other EJBs, Enterprise Application Clients and Web Services Clients when exposed as Web services. Other benefits include scalability because the EJB container can easily be tuned and scaled up when load increases.

Perform the following operations to create this Enterprise Java Bean:

1. From the `Projects` window, right-click on the `JavaEEDBFaculty-ejb` project and select the `New > Session Bean` menu item.
2. In the opened `New Session Bean` dialog, specify the `EJB Name` as `FacultySession`, the `Package` as `com.javaeedbfaculty.ejb`, the `Session Type` as `Stateless` and leave two `Create Interface` checkboxes unchecked. Your finished `New Session Bean` dialog box should match one that is shown in Figure K.6. Click on the `Finish` button to complete this creation of Session Bean process.

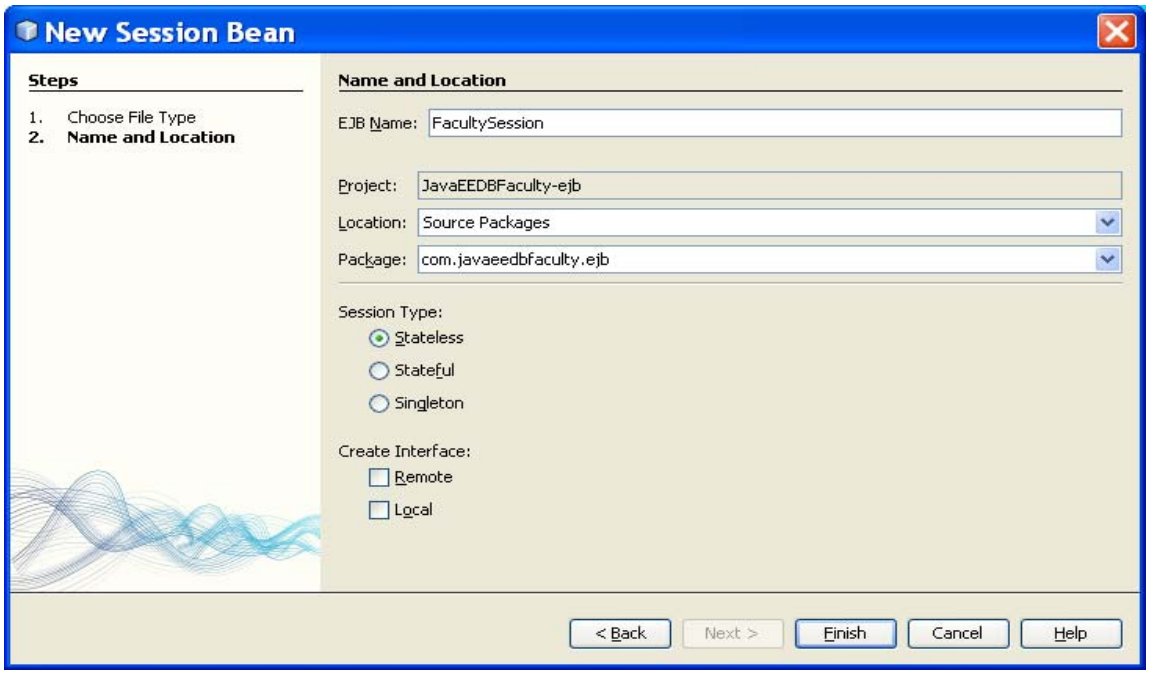

Figure K.6 The finished New Session bean dialog box.

3. From the `Projects` window, navigate to the source of the newly created `Session Bean` (skeleton) by double clicking on the `FacultySession` item that is under the `Enterprise Beans` folder, as shown in Figure K.7.
4. In the opened code window, right-click in any place in this window and select the `Persistence > Use Entity Manager` menu item from the popup menu, and then you can find that the `@PersistenceContext` notation is inserted automatically into this code window, so now the `EntityManager`, with variable name `em`, is ready to be used. The auto-created codes by the NetBeans have been highlighted in bold and shown in Figure K.8.

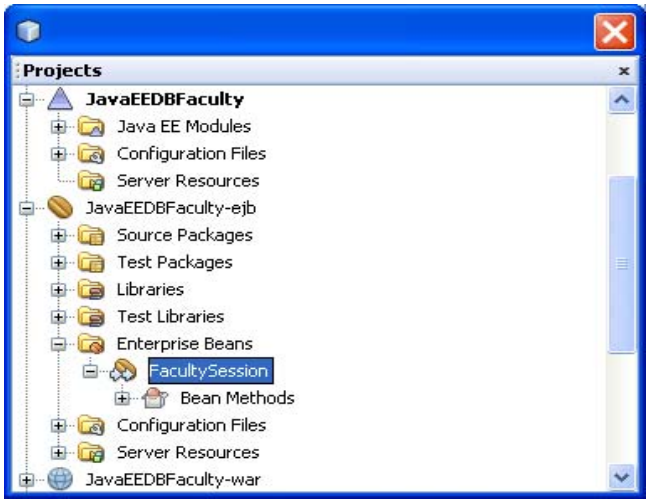

Figure K.7 The new created FacultySession Bean.

5. Create a business method for the Session Bean: **Retrieve**() since we need to use this method to perform data query from the Faculty table later; right-click in the `Insert Code > Add Business Method` section in the code window, and select the `Insert Code` menu item from the popup menu, under the `Generate` list, select the `Add Business Method` menu item.

```
package com.javaeedbfaculty.ejb;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
@LocalBean
public class FacultySession {
    @PersistenceContext(unitName = "JavaEEDBFaculty-ejbPU")
    private EntityManager  em;

    public void persist(Object object) {
        em.persist(object);
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")
}
```

Figure K.8 The inserted codes for the Entity Manager.

6. In the opened `Add Business Method` dialog, provide `Retrieve` to the `Name` field as the name of this method. Click on the `Browse` button that is next to the `Return Type` combo box and type the `list` on the `List Name` field from the `Find Type` dialog to scan the available type list. Select the item `List(java.util)` from the list and click on the `OK` button in the `Find Type` dialog to select this type. Your finished `Add Business Method` dialog should match one that is shown in Figure K.9.

Click on the `OK` button to close this adding method process.

Now let's develop the codes for this `Retrieve()` methods to implement the intended function. Edit this method by adding the codes that are shown in Figure K.10 into this method.

The edited codes have been highlighted in bold, and let's have a closer look at this piece of codes to see how it works.

A. Inside the `Retrieve()` method, first we create a Java Persistence API query instance `query` and execute a named or static query to pick up all columns from the `Faculty` entity. The query result is returned and stored to the `query` instance.
B. The `getResultList()` method is executed to get the query result and return it to the `List` object.

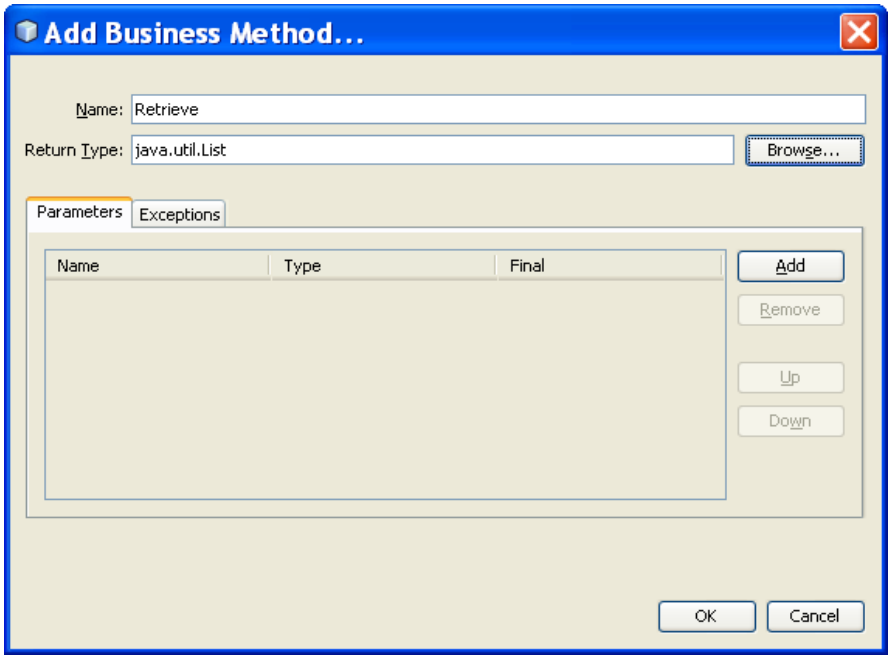

Figure K.9 The finished Add Business Method dialog box.

```
@Stateless
@LocalBean
public class FacultySession {
    @PersistenceContext(unitName = "JavaEEDBFaculty-ejbPU")
    private EntityManager em;

    public void persist(Object object) {
        em.persist(object);
    }

    public List<Faculty> Retrieve() {
A       Query  query = em.createNamedQuery("Faculty.findAll");
B       return  query.getResultList();
    }
```

Figure K.10 The edited coding for both business methods.

After you finish adding this piece of codes into the `Retrieve()` method, you may encounter some in-time compiling errors for some class and interface, such as the `Faculty` class and `Query`

interface. The reason for that is because those classes and interfaces are defined in the different packages and you need to involve those packages into this project file. Perform the following import operations to add those packages to the top of this project file:

```
import javax.persistence.Query;
import com.javaeedbfaculty.entity.Faculty;
```

Your complete code window for this CustomerSession class file should match one that is shown in Figure K.11. The new inserted codes have been highlighted in bold.

```
package com.javaeedbfaculty.ejb;

import java.util.List;
import javax.ejb.Stateless;
import javax.ejb.LocalBean;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;
import com.javaeedbfaculty.entity.Faculty;

@Stateless
@LocalBean
public class FacultySession {
    @PersistenceContext(unitName = "JavaEEDBFaculty-ejbPU")
    private EntityManager em;

    public void persist(Object object) {
        em.persist(object);
    }

    public List<Faculty> Retrieve() {
        Query query = em.createNamedQuery("Faculty.findAll");
        return query.getResultList();
    }

    // Add business logic below. (Right-click in editor and choose
    // "Insert Code > Add Business Method")

}
```

Figure K.11 The complete codes for the FacultySession class.

Now you can build and compile the project files so far we have developed by clicking on the `Clean and Build Main Project` button. Up to this point, we have completed the tasks required to be done in the EJB project, and we will move on to the next tier, JSF pages.

K.4 Using JavaServer Faces (JSF) 2.0

Before we can create the Web pages for this project, ensure that the JavaServer Faces framework is added to the Web project, `JavaEEDBFaculty-war`. Perform the following operations to confirm this addition.

1. In the `Projects` window, right-click on the Web project, `JavaEEDBFaculty-war`, and select the `Properties` menu item from the popup menu.
2. Under the `Categories` items, select `Frameworks`, and ensure that the `JavaServer Faces` has been added into the `Used Frameworks` list. If not, click on the `Add` button to open the `Add a Framework` dialog to add the `JavaServer Faces` to the project by

selecting it and clicking on the OK button. Your finished Project Properties window should match one that is shown in Figure K.12. Click on the OK button to complete this confirmation process.

Now we need to create the JSF pages to present the screens to perform the **R**ead function. To achieve this, we will be creating 2 Web pages:

- **FacultyList** – listing of all Faculty records in our sample database in a tabular form
- **FacultyDetails** – view/edit the details of the selected Faculty record

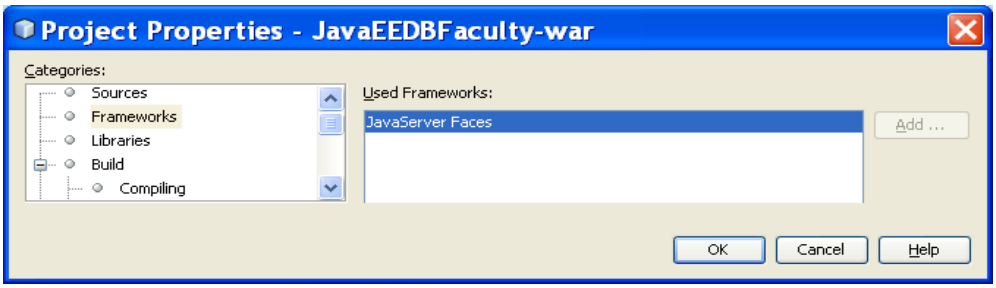

Figure K.12 The finished Project Properties window.

However, before creating the JSF pages, we first need to create the managed bean that will be providing the required services for the JSF pages that will be created later.

K.5 Creating the Faculty Managed Bean

Perform the following operations to create the managed bean that provides message communications between the Web pages and the Java persistence API.

1. In the Projects window, right-click on the Web project, JavaEEDBFaculty-war, and select the New > JSF Managed Bean item by clicking on it to open the New JSF Managed Bean dialog.
2. Specify the FacultyMBean as the Class Name, and com.javaeedbfaculty.web as the Package Name, faculty as the Name, and the Scope to be session. Your finished New JSF Managed Bean dialog should match one that is shown in Figure K.13. Click on the Finish button to complete this creation of a new JSF managed bean process.
3. Open the code window of the newly created class, FacultyMBean.java, by double clicking on this file folder in the Projects window, right-click inside the constructor of this class and select the Insert Code menu item, and select the Call Enterprise Bean item under the Generate list.
4. In the opened Call Enterprise Bean dialog, expand the JavaEEDBFaculty-ejb project and select the FacultySession and select the No Interface option. Also disable the Local and Remote options because we created the Session Bean with no interface for Referenced Interface, and then click on the OK button.
5. Notice the automatically generated variable, facultySession, which represents an instance of the session bean, at the beginning of the class declaration.

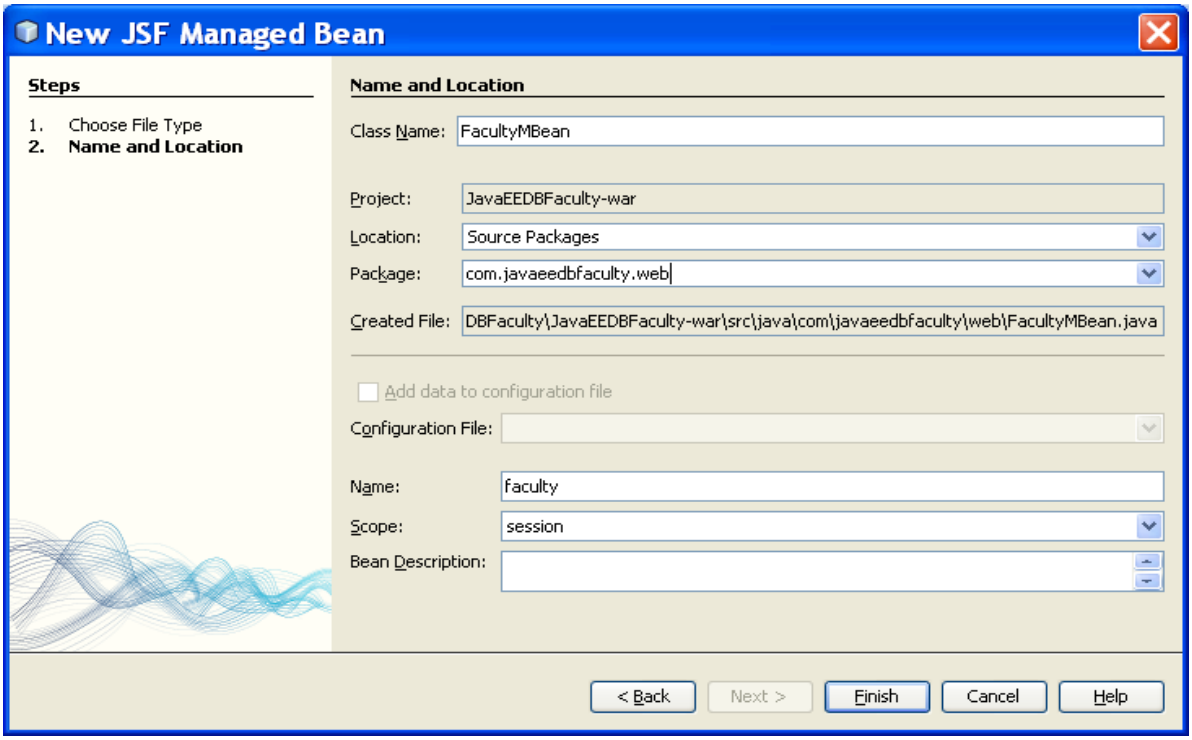

Figure K.13 The finished New JSF Managed Bean dialog.

Now let's do the coding jobs for this class file.

First we need to add the following import packages statements into this class to enable the compiler to correctly locate and identify related objects we will use in this class:

```
import com.javaeedbfaculty.entity.Faculty;
import java.util.List;
```

Second let's add the rest of the methods, properties and action handlers, and its implementations to the class as shown in Figure K.14, which will be used by the JSF pages later. The new added codes have been highlighted in bold.

Let's have a closer look at this new added piece of codes to see how it works.

A. Two packages have been added into this class file since we need to use the `Faculty` entity and the `List` class in this file and both of them are defined in those two different packages.

B. In order to use the `Faculty` entity to access the `Faculty` table in our sample database, we need to create a new instance of this class, `faculty`.

C. The `getFaculties()`method is defined to pick up a list of faculty objects to be displayed in the data table. Exactly the `Retrieve()`method defined in our `FacultySession` bean will be executed to perform this retrieving operation.

D. The `getDetails()`method is defined to return the selected `Faculty` object.

E. The `showDetails()`method is exactly a handler to handle the users' selection from the list.

F. The `list()`method is a event handler used to direct this event to open the Faculty List page we will create in the next section.

At this point, we have finished editing and modifying the codes for our `Faculty Managed Bean` code window. Your finished code window should match one that is shown in Figure K.14.

```
   package com.javaeedbfaculty.web;

   import com.javaeedbfaculty.ejb.FacultySession;
   import javax.ejb.EJB;
   import javax.faces.bean.ManagedBean;
   import javax.faces.bean.SessionScoped;
A  import com.javaeedbfaculty.entity.Faculty;
   import java.util.List;

   @ManagedBean(name="faculty")
   @SessionScoped
   public class FacultyMBean {
      @EJB

      private  FacultySession  facultySession;
B     private  Faculty  faculty;

      /** Creates a new instance of FacultyMBean */
      public  FacultyMBean() {

      }
      /* Returns list of faculty objects to be displayed in the data table */
C     public  List<Faculty>  getFaculties() {
         return  facultySession.Retrieve();
      }

      /* Returns the selected Customer object */
D     public  Faculty  getDetails() {
         //Can either do this for simplicity or fetch the details again from the
         //database using the Faculty ID
         return  faculty;
      }

      /* Action handler - user selects a faculty record from the list */
E     public  String  showDetails(Faculty  faculty) {
         this.faculty = faculty;
         return  "DETAILS";
      }

      /* Action handler - goes to the Customer listing page */
F     public  String  list(){
         System.out.println("###LIST###");
         return  "LIST";
      }
   }
```

Figure K.14 The modified Faculty Managed Bean code window.

Now, let's create the first web page that lists the Faculty records in the database in a tabular form.

K.6 Creating the Faculty Listing Web Page

Perform the following operations to create this Faculty Listing Web page:

1. In the `Projects` window, right-click on our Web project, `JavaEEDBFaculty-war`, and select the `New > JSF Page` item from the popup menu. On the opened the `New JSF`

File dialog, specify FacultyList as the File Name and check the JSP File radio button under the Options group. Your finished New JSF File dialog should match one that is shown in Figure K.15. Click on the Finish button to continue.

2. In the opened code window, drag the item, JSF Data Table from Entity from the Palette window and drop it in between the <body> </body> tags of the newly generated file, FacultyList.jsp, as shown in Figure K.16. If the Palette window is not opened, go to the Window menu item and click on the Palette item to open it.

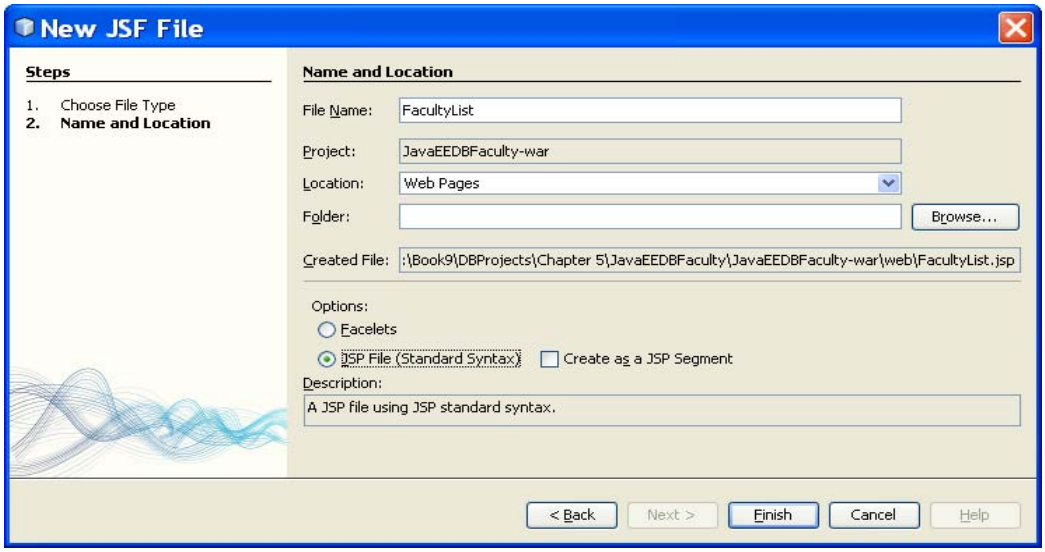

Figure K.15 The finished New JSF File dialog box.

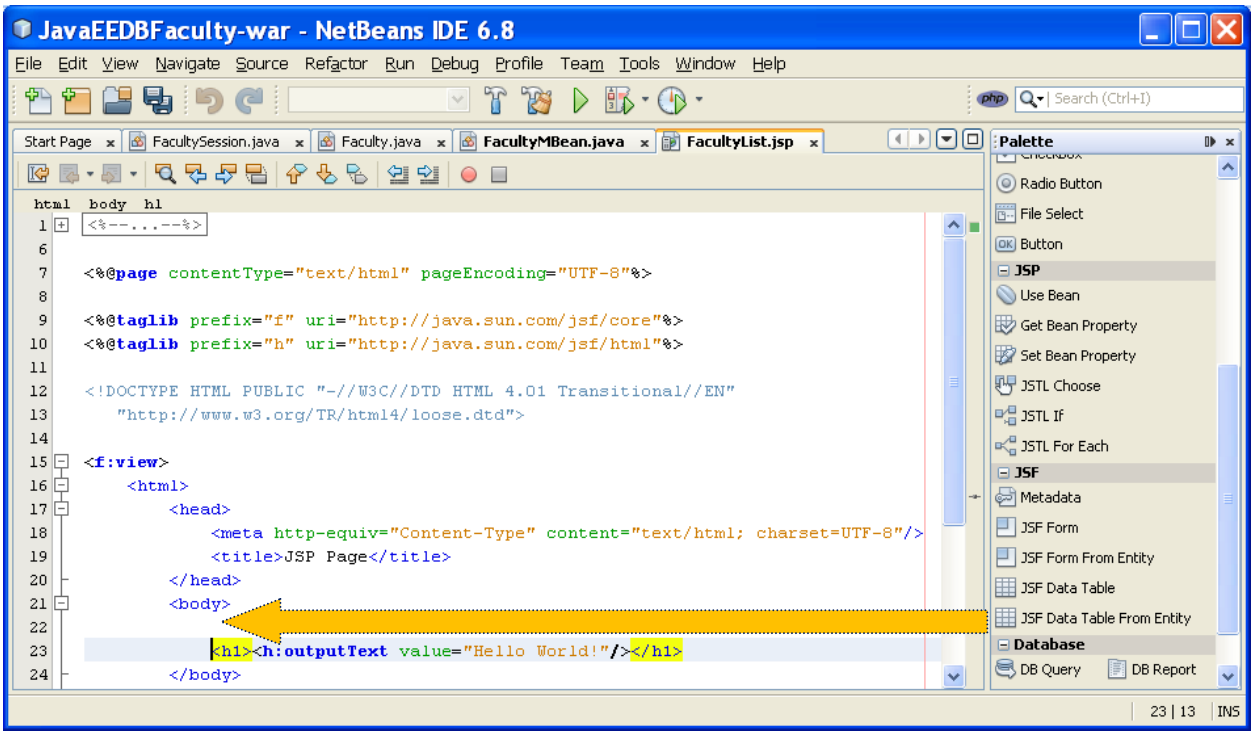

Figure K.16 The inserted JSF Data Table from Entity item.

You can use this dragged `JSF Data Table from Entity` item to replace the original instruction:

```
<h1><h:outputText value="Hello World!"/></h1>
```

Or you can leave the original instruction at the bottom of this new inserted item.

3. A dialog with the title, `JSF Table from Entity` appears; from the `Entity` combo box, select the `com.javaeedbfaculty.entity.Faculty` as the `Entity Bean`, and the `faculty.faculties` as the `Managed Bean Property`, as shown in Figure K.17, and click on the `OK` button.

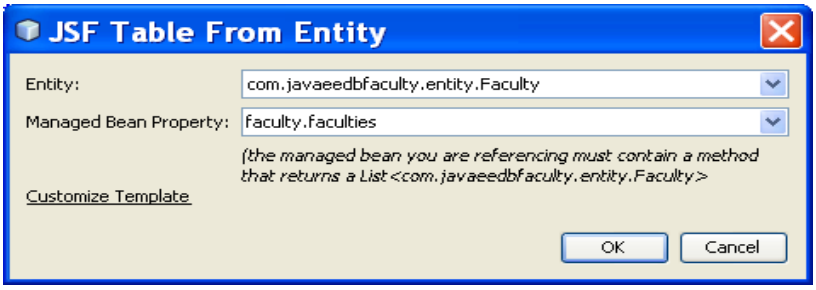

Figure K.17 The finished JSF Table From Entity dialog box.

Notice that the results of this operation are lines of codes automatically generated to display a default list of the Faculty objects.

At this point, we are ready to see the result of the first web page created so far. However, before we can do that, first we need to add the jdbc driver for Microsoft SQL Server database to our project to enable the project to know the location of this driver as the project runs.

K.7 Add the JDBC Driver for the SQL Server Database into the Project

To enable our Java EE 6 project to load and connect to our sample SQL Server database, CSE_DEPT we built in Chapter 2, we need to:

1. Download this driver from the site http://msdn.microsoft.com/data/jdbc/
2. Configure the TCP/IP protocol and setup for the SQL server
3. Setup a SQL Server database connection using NetBeans 6.8 IDE

Refer to section 6.2.1.2 in Chapter 6 to complete these steps.

When we finished downloading this jdbc driver **sqljdbc4.jar**, the default location of this driver is at: **C:\Program Files\Microsoft SQL Server JDBC Driver 2.0|sqljdbc_2.0|enu**. To add this jdbc driver to our project `JavaEEDBFaculty`, right click on the Web project `JavaEEDBFaculty-war` and select the `Properties` item from the popup menu. On the opened `Project Properties` dialog, select the `Libraries` item from the `Categories` list. Then click on the `Add JAR/Folder` button to open the `Add JAR/Folder` dialog.

Click on the dropdown arrow in the `Look in` combo box and browse to the location where our downloaded jdbc driver **sqljdbc4.jar** is located, which is **C:\Program Files\Microsoft SQL Server**

JDBC Driver 2.0 | sqljdbc_2.0 | enu. Then select this driver and click on the `Open` button. Your finished `Project Properties` dialog box should match one that is shown in Figure K.18.

Click on the `OK` button to complete this adding process.

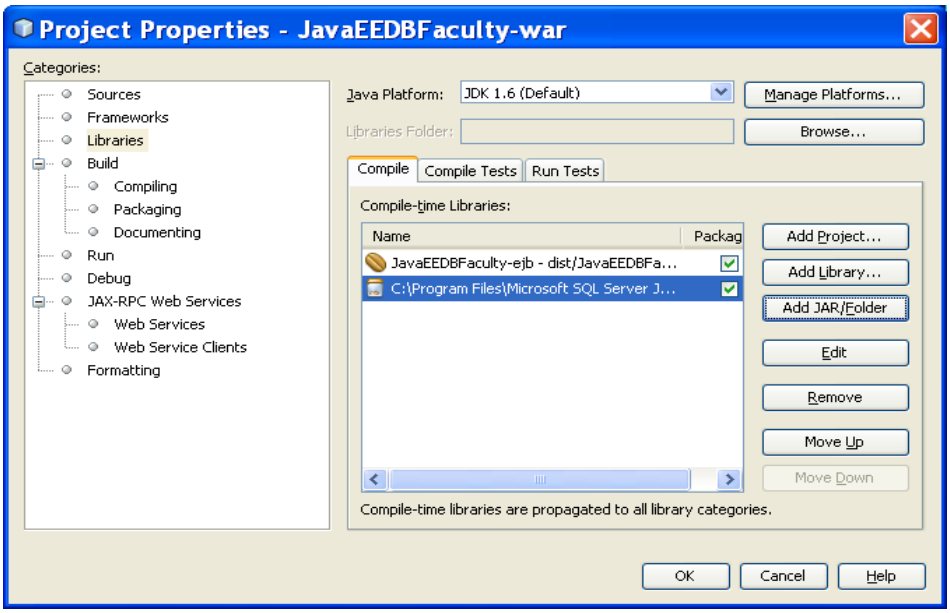

Figure K.18 The finished Project Properties dialog box.

Now we are ready to build and run our first JSP Web page, `FacultyList.jsp`.

K.8 Building and Running the First Java EE 6 Web Page

Perform the following operations to build and run this JSP Web page:

1. In the `Projects` window, right-click on our `JavaEEDBFaculty` project and select the `Clean and Build` menu item from the popup menu to build our project. If everything is fine, right click on our project `JavaEEDBFaculty` again and select `Deploy`. Enter the username and password you used when you installed the Java Glassfish v3 Server in section 5.3.5.2.1 in Chapter 5 to the `Authentication Required` dialog box if it is displayed. In this application, we used `admin` and `reback` as the username and password for the Java Glassfish v3 Server in this installation.
2. To confirm that the deployment is successful, navigate to the `Applications` folder in the Glassfish server under the `Services` view, as shown in Figure K.19, and check if the application `JavaEEDBFaculty` exists.

Now open the browser and go to URL: http://localhost:8082/JavaEEDBFaculty-war/faces/FacultyList.jsp and you should see the opened `Faculty` data table in the sample database, which is shown in Figure K.20. The port we have used for our Glassfish v3 server is 8082 since the default port 8080 has been occupied in the current machine.

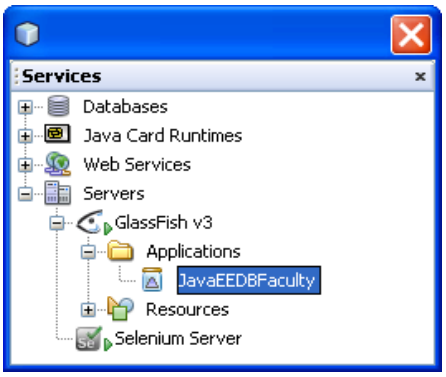

Figure K.19 The deployed JavaEEDBFaculty project.

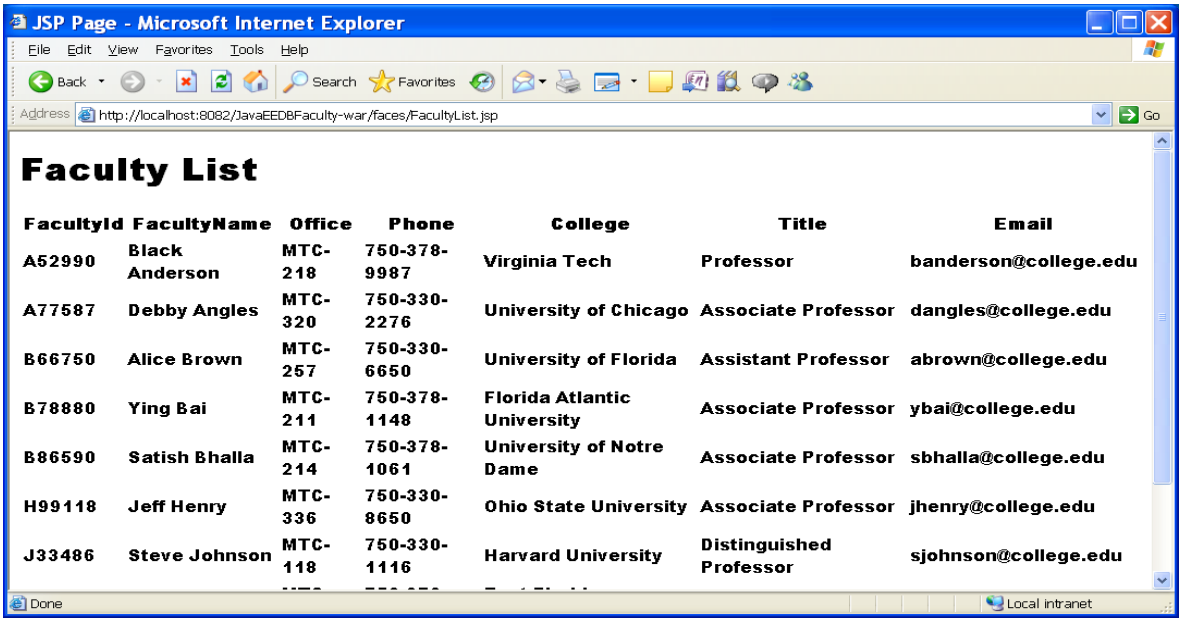

Figure K.20 The opened Web page contained the Faculty data table.

One issue to be noted here is: the screen is very raw and without any beautification. We will discuss this issue and find solutions in the following section. This means that we will build a `FacultyDetails` JSF page to display the selected faculty information based on the `facyltyId`.

K.9 Creating the Faculty Details Web Page

Now we will handle the issue to create the page where the details of the selected faculty are displayed. Perform the following operations to complete this displaying:

1. In the `Projects` window, right-click on the Web project, `JavaEEDBFaculty-war`, and select `New > JSF Page`, specify `FacultyDetails` as the `File Name` and `JSP File` under the `Options`. Click on the `Finish` button to complete this process.
2. In the code window, drag the item, `JSF Form from Entity` from the `Palette` window and drop it in between the <body> </body> tags of the newly generated file, `FacultyDetails.jsp`, as shown in Figure K.21.

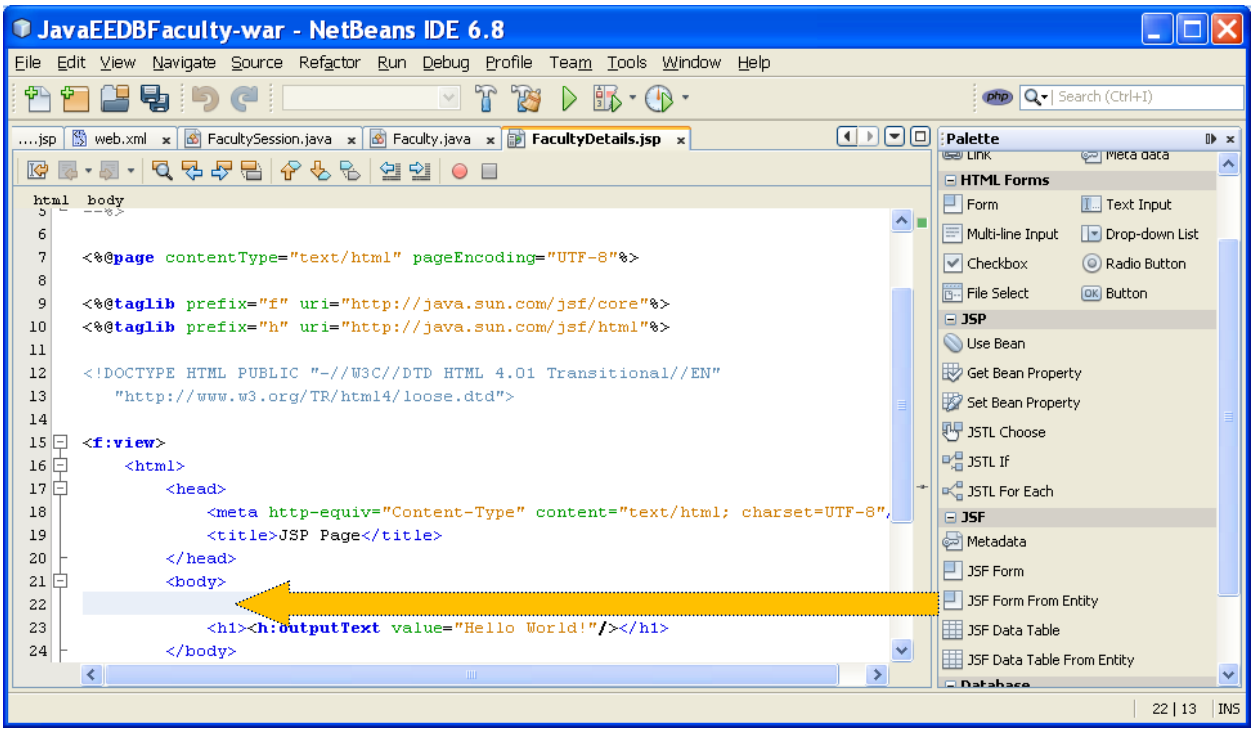

Figure K.21 Drag the JSF Form From Entity from the Palette window.

3. In the `JSF Form From Entity` dialog, select our Faculty entity class file `com.javaeedbfaculty.entity.Faculty` from the `Entity` combo box and `faculty.details` from the `Managed Bean Property` combo box, as shown in Figure K.22. Click on the `OK` button to complete this process. Notice the result of this are lines of codes automatically generated to display label and input field of all the attributes in Faculty object in a 2 columns grid.

To enable the navigation from the `Listing` page to the `Details` and vice versa, we need to create and edit the `faces-config.xml` with the `PageFlow` editor and connect these 2 pages together.

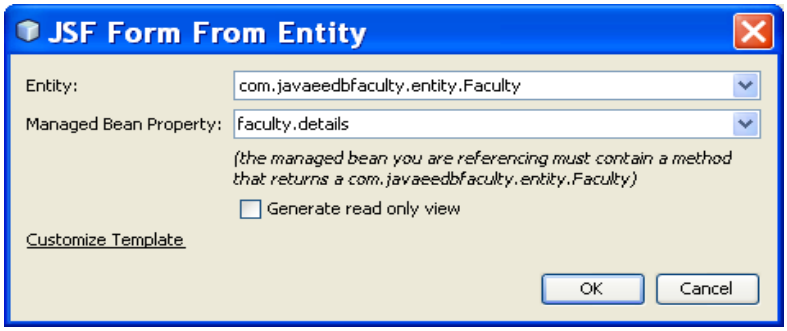

Figure K.22 The finished JSF Form From Entity dialog box.

First let's have a clear picture and idea about the faces-config.xml file and the PageFlow editor.

K.10 Creating and Editing the faces-config.xml Configuration File

When you create a new Java EE Web application with JSF, the JSF also creates some configuration files, and all Web-related and JSF-related components are included in the following two configuration files:

- web.xml – Contains general Web application configuration file.
- faces-config.xml - Contains the configuration of the JSF application.

The detailed functions for these two configuration files are:

- **web.xml:** JSF requires the central configuration list web.xml in the directory WEB-INF of the application. This is similar to other web-applications which are based on Servlets. You must specify in web.xml that a FacesServlet is responsible for handling JSF applications. FacesServlet is the central controller for the JSF application and it receives all requests for the JSF application and initializes the JSF components before the JSP is displayed.
- **faces-config.xml:** The faces-config.xml file allows the JSF to configure the application, managed beans, convertors, validators, and navigation.

The NetBeans IDE provides two distinct *views* for the faces-config.xml file: the XML View, which displays the XML source code, and the PageFlow view, which is a graphical interface that depicts JSF navigation rules defined in the faces-config.xml file.

The PageFlow view displays the navigation relationships between JSF pages, indicating that a navigation from one JSF page to another JSF page occurs when response is passed to JSF's NavigationHandler.

Double-clicking on components in the PageFlow view enables you to navigate directly to the source file. Likewise, if you double-click on the arrow between the two components, the editor will focus on the navigation rule defined in the faces-config.xml XML view.

Now let's first create a faces-config.xml file for our JavaEEDBFaculty application. Perform the following operations to create this configuration file:

1. Right click on our Web application JavaEEDBFaculty-war and select the New > Other item from the popup menu to open the New File dialog.
2. Select the JavaServer Faces from the Categories list and JSF Faces Configuration from the File Types list, as shown in Figure K.23. Click on the Next button to continue.
3. In the opened New JSF Faces Configuration dialog, enter faces-config into the File Name field as the name of this file, and click on the Finish button.

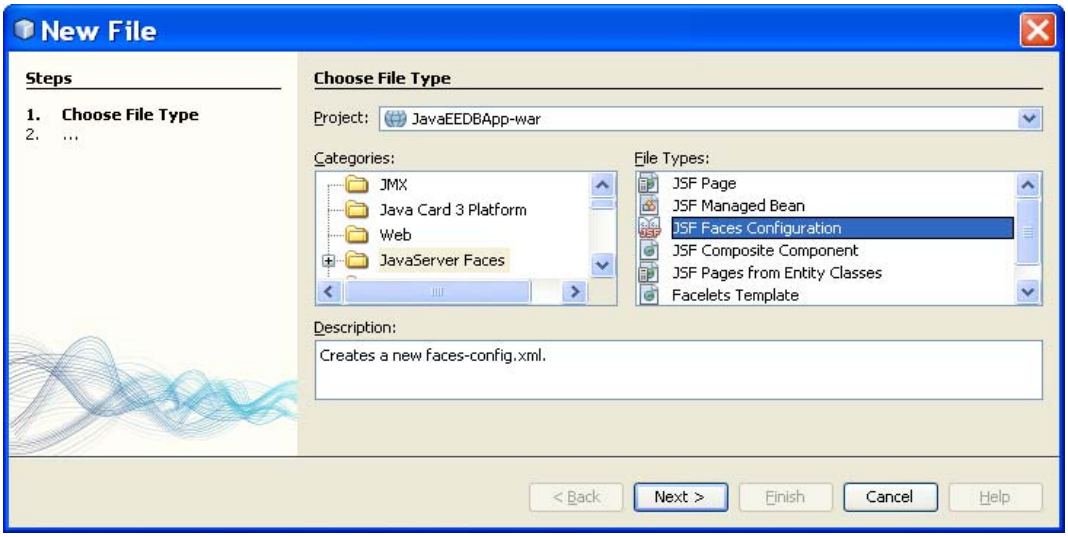

Figure K.23 The New File dialog box.

The new created `faces-config.xml` file is opened and shown in Figure K.24.

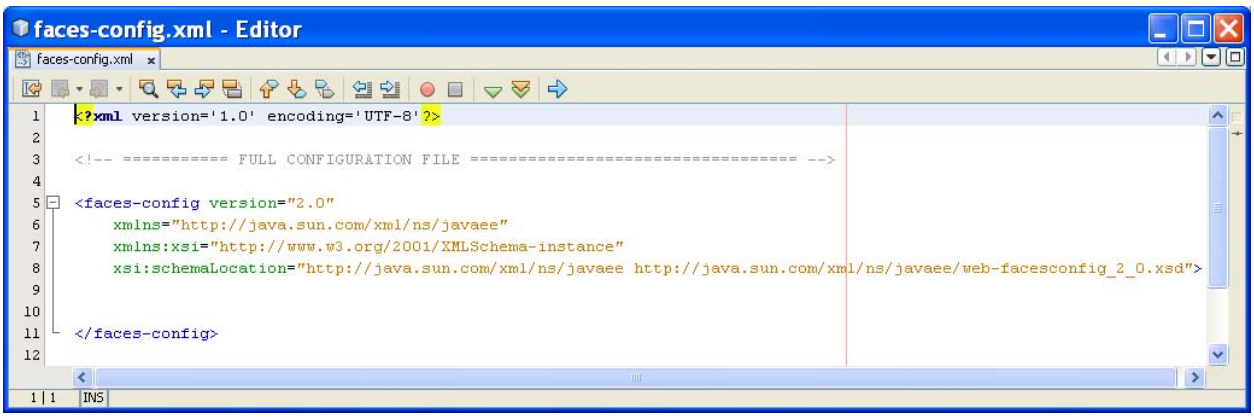

Figure K.24 The created faces-config.xml file.

First let's add our managed bean into this `faces-config.xml` file by perform the following operations:

1. Right click on any location inside the opened `faces-config.xml` file, and select `Insert > Managed Bean` item from the popup menu. In some cases, you may need to close and re-open the NetBeans IDE to have this `Insert` menu item available.
2. In the opened `Add Managed Bean` dialog, enter `faculty` into the `Bean Name` field, and click the `Browse` button to open the `Find Type` dialog. Type `Faculty` to the `Type Name` field and select the `FacultyMBean` item from the list, and click on the `OK` button.
3. Select the `session` from the `Scope` combo box.
4. You can enter some description for this managed bean into the `Bean Description` box if you like.

Your finished `Add Managed Bean` dialog box should match one that is shown in Figure K.25.

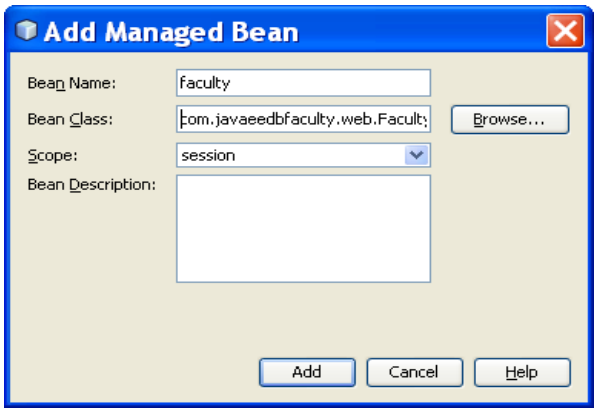

Figure K.25 The finished Add Managed Bean dialog box.

Click on the Add button to add this `FacultyMBean` as our managed bean into this configuration file. Now click on the `XML` tag to open this configuration file in the XML view, and you can find that the related XML tags have been added into this `faces-config.xml` file, which is shown in Figure K.26. The new added codes have been highlighted in bold.

```
<?xml version='1.0' encoding='UTF-8'?>

<!-- ========== FULL CONFIGURATION FILE ================================== -->

<faces-config version="2.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">

  <managed-bean>
A     <managed-bean-name>faculty</managed-bean-name>
B     <managed-bean-class>com.javaeedbfaculty.web.FacultyMBean</managed-bean-class>
C     <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

</faces-config>
```

Figure K.26 The added managed bean to the faces-config.xml file.

Let's have a closer look at this piece of new added codes to see how it works.

A. The name of our managed bean, `faculty`, is added to the managed-bean-name tag.
B. The class of our managed bean, `FacultyMBean`, with its namespace, is added under the `managed-bean-class` tag to indicate the managed bean class used in this application.
C. The scope of this managed bean is session and has been added under the managed-bean-scope tag.

Note: This faces-config.xml **file may be located at two locations: 1) under the** Web Pages\WEB-INF **folder, and 2) under the** Configuration Files **folder. It is the same file that just resides at the different locations.**

To setup navigation relationship between JSF pages, especially between the `FacultyList` and `FacultyDetails` pages in this application, we need to edit this configuration file using the `PageFlow` editor.

Now click on the `Clean and Build Main Project` button to build our modified project to cover the new added `faces-config.xml` file.

Then double click on our new added and edited `faces-config.xml` file from either location to open this file. Click on the `PageFlow` button to open its page flow view.

Note: In some cases, you may need to close and restart the NetBeans to make this new added faces-config.xml **to have the PageFlow view.**

In the opened `PageFlow` view, totally there are four JSF pages are displayed in this configuration view: `index.jsp`, `index.xhtml`, `FacultyList.jsp` and `FacultyDetails.jsp`, as shown in Figure K.27.

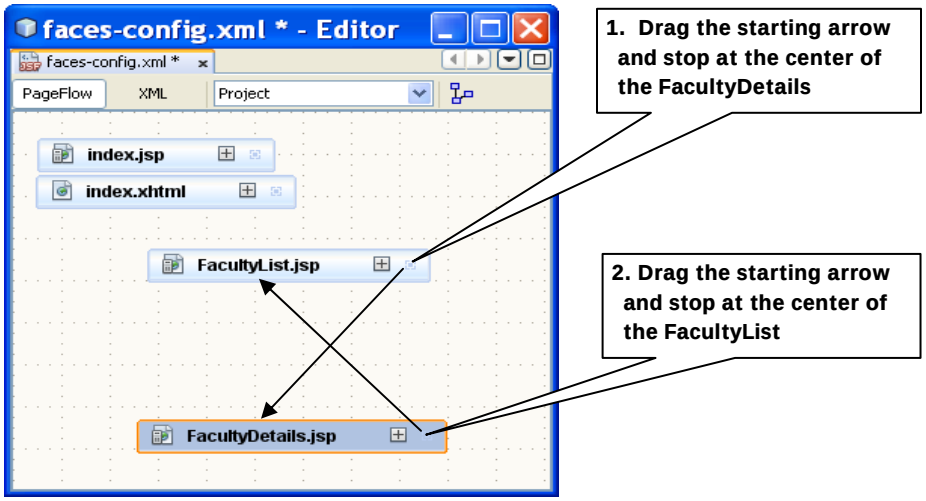

Figure K.27 The opened PageFlow view of the faces-config.xml file.

To setup the navigation relationships between the `FacultyList` and `FacultyDetails` JSF pages, perform the following operations:

1. Move your cursor to the starting arrow location as shown in Figure K.27 until a square appears in the `FacultyList.jsp` page object. Then click on this square and drag this stating arrow and point to and stop at the center of the `FacultyDetails.jsp`, as shown in Figure K.27 box-1. A navigation link is established with the default name `case1`, as shown in Figure K.28.
2. Double click on the default navigation link `case1` and change its name to `DETAILS`.
3. Perform a similar operation to create another navigation link from the `FacultyDetails` to the `FacultyList`, as shown in Figure K.27 box-2.

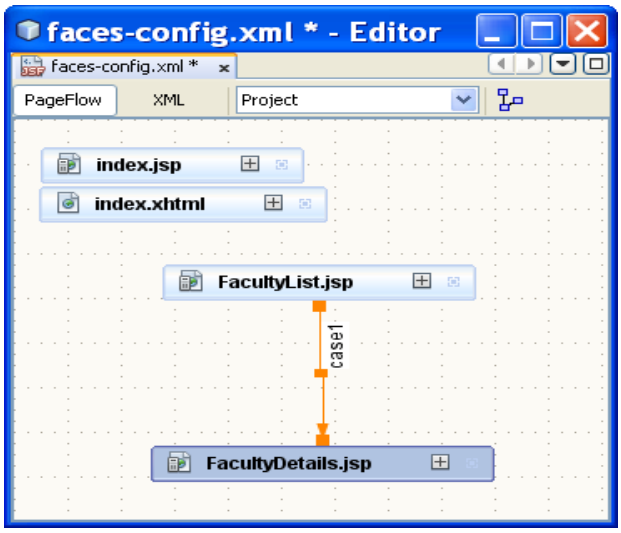

Figure K.28 The established default navigation link.

4. Double click on the new established link and change its name to `LIST`. Your finished PageFlow view of two JSF page objects should match one that is shown in Figure K.29.

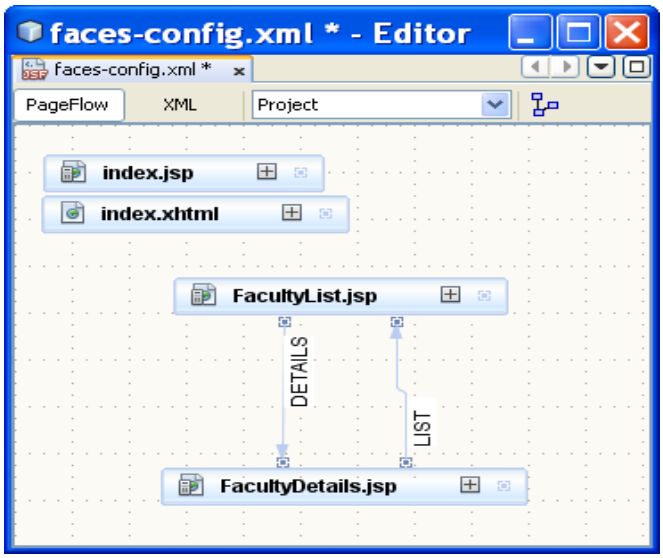

Figure K.29 The finished PageFlow view of the JSF page objects.

Now if you click on the `XML` button to open the XML view of this `faces-config.xml` file, you can find that the navigation rules shown in Figure K.30 have been added into this file. The new added codes have been highlighted in bold.

As shown in Figure K.30, two navigation rules, which are indicted by **A** and **B**, have been added into this configuration file. The first one is from the `FacultyList` to the `FacultyDetails` and the second is from the `FacultyDetails` to the `FacultyList`.

Notes: the <from-outcome> strings LIST **and** DETAILS **must match the return String of the** list() **and** showDetails() **methods defined in the FacultyMBean.**

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- ============ FULL CONFIGURATION FILE ================================== -->
<faces-config version="2.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">
  <!-- a normal Managed Bean -->
   <managed-bean>
      <managed-bean-name>CustomerMBean</managed-bean-name>
      <managed-bean-class>view.CustomerMBean</managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
A  <navigation-rule>
     <from-view-id>/FacultyList.jsp</from-view-id>
     <navigation-case>
        <from-outcome>DETAILS</from-outcome>
        <to-view-id>/FacultyDetails.jsp</to-view-id>
     </navigation-case>
   </navigation-rule>
B  <navigation-rule>
     <from-view-id>/FacultyDetails.jsp</from-view-id>
     <navigation-case>
        <from-outcome>LIST</from-outcome>
        <to-view-id>/FacultyList.jsp</to-view-id>
     </navigation-case>
   </navigation-rule>
</faces-config>
```

Figure K.30 The new added navigation rules.

K.11 Editing the General Web Application Configuration File web.xml

As we mentioned in the last section, the `web.xml` file contains the central configuration list including all configuration descriptions about the JSF pages built in the project. To include our new added `faces-config.xml` configuration file into our project, we need to add some XML tags to this `web.xml` file to enable system to know that a new edited `faces-config.xml` file has been added and will be implemented in this project.

Perform the following operations to complete this addition process:

1. Open the `web.xml` file by double clicking on it from the `Projects` window. Regularly this file should be located at the `WEB-INF` folder or the `Configuration Files` folder.
2. On the opened `web.xml` file, add the XML tags that are shown in Figure K.31 into this configuration file. The new added XML tags have been highlighted in bold.

Your completed `web.xml` configuration file should match one that is shown in Figure K.31

To setup a relationship between our two JSF pages, `FacultyList.jsp` and `FacultyDetails.jsp`, and enable a switching between these two pages, we need to add some JSF tags to both page files.

Next let's modify our `FacultyList.jsp` page to setup a connection relationship with our `FacultyDetails.jsp` page.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.CONFIG_FILES</param-name>
        <param-value>/WEB-INF/faces-config.xml</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Figure K.31 The modified general Web configuration file web.xml.

K.12 Modifying the FacultyList and FacultyDetails Pages to Perform Page Switching

In the last section, we have established the navigation relationships between the `FacultyList` and the `FacultyDetails` JSF pages using the navigation rules in the `faces-config.xml` file. In order to trigger those rules and switch from the `FacultyList` to the `FacultyDetails` page, we need to modify some part of the `FacultyList` page to accomplish this navigation. We want to use the `facultyID` as a connection key or link to display the detailed record for only one faculty record based on the `facultyID`.

To do this modification, open the `FacultyList.jsp` page from the `Projects` window and replace the coding line

```
<h:outputText value="#{item. facultyId}"/>
```

With the following coding lines

```
<h:commandLink  action="#{faculty.showDetails(item)}"
                              value="#{item.facultyId}"/>
```

Your modified part on the `FacultyList.jsp` is shown in Figure K.32.

```
<f:view>
   <html>
      <head>
         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
         <title>JSP Page</title>
      </head>
      <body>
         <h:form>
            <h1><h:outputText value="Faculty Listing"
            <h:dataTable value="#{faculty.faculties}" var="item"
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="Faculty
                  </f:facet>
                  <h:outputText value="#{item.facultyId}"/>
               </h:column>
               <h:column>
                  <f:facet name="header">
                     <h:outputText value="FacultyName"/>
                  </f:facet>
                  <h:outputText value="#{item.facultyName}"/>
               </h:column>
               ..........
```

Using the following codes to replace the outputText to navigate to the FacultyDetails page

```
<h:commandLink action="#{faculty.showDetails(item)}"
               value="#{item.facultyId}"/>
```

Figure K.32 The modified codes in the FacultyList.jsp page.

Next open the `FacultyDetails.jsp` page and add one JSF tag to the end of this page, as shown in Figure K.33. The new added tag has been highlighted in bold. This command button will enable users to switch back from the `FacultyDetails` page to the `FacultyList` page, and it has a similar function as that of `Back` button on a Web browser.

```
               ..........
               <h:outputLabel value="Title:" for="title" />
               <h:inputTextarea rows="4" cols="30" id="title" value="#{faculty.details.title}" title="Title" />
               <h:outputLabel value="Email:" for="email" />
               <h:inputTextarea rows="4" cols="30" id="email" value="#{faculty.details.email}" title="Email" />
            </h:panelGrid>
            <h:commandButton id="list" value="List" action="#{faculty.list}" />
         </h:form>
         <h1><h:outputText value="Hello World!"/></h1>
      </body>
   </html>
</f:view>
```

Figure K.33 The modified codes for the FacultyDetails page.

Now we can build and run the project to test the functions of this project.

K.13 Building and Running the Entire Java EE 6 Project

At this point, we have completed all coding jobs for this project. To see the running result, first let's build the application by right clicking on our project `JavaEEDBFaculty` and select the `Clean and Build` item, and deploy the application by right clicking on our project `JavaEEDBFaculty` and select the `Deploy` item.

If everything is fine, open a Web browser and go to the Customer listing page at URL, http://localhost:8082/JavaEEDBFaculty-war/faces/FacultyList.jsp. You can find that all faculty IDs have been underscored. Click on the `Faculty ID` on the first row in the table to open the `FacultyDetails` page to query and display the detailed record for this faculty ID only.

A running result of this project with a faculty ID of B78880 is shown in Figure K.34. You can click on the `List` button to return to the `FacultyList` page and re-select some other facultyID to see more results.

A complete Java EE 6 Database-related project `JavaEEDBFaculty` can be found from the folder `DBProjects\Chapter 5` that is located at the site: `ftp://ftp.wiley.isbn/JavaDB`.

You can download this project from that site and run on your computer. However, you have to make sure that you have installed all required software before you can run this project on your computer:

1. Java Enterprise Edition 6
2. Glassfish v3
3. NetBeans 6.8 IDE or higher version of IDE.
4. The JDBC Driver for SQL Server database has been installed in your computer and has been added into the Web page `JavaEEDBFaculty-war` as a JAR file.

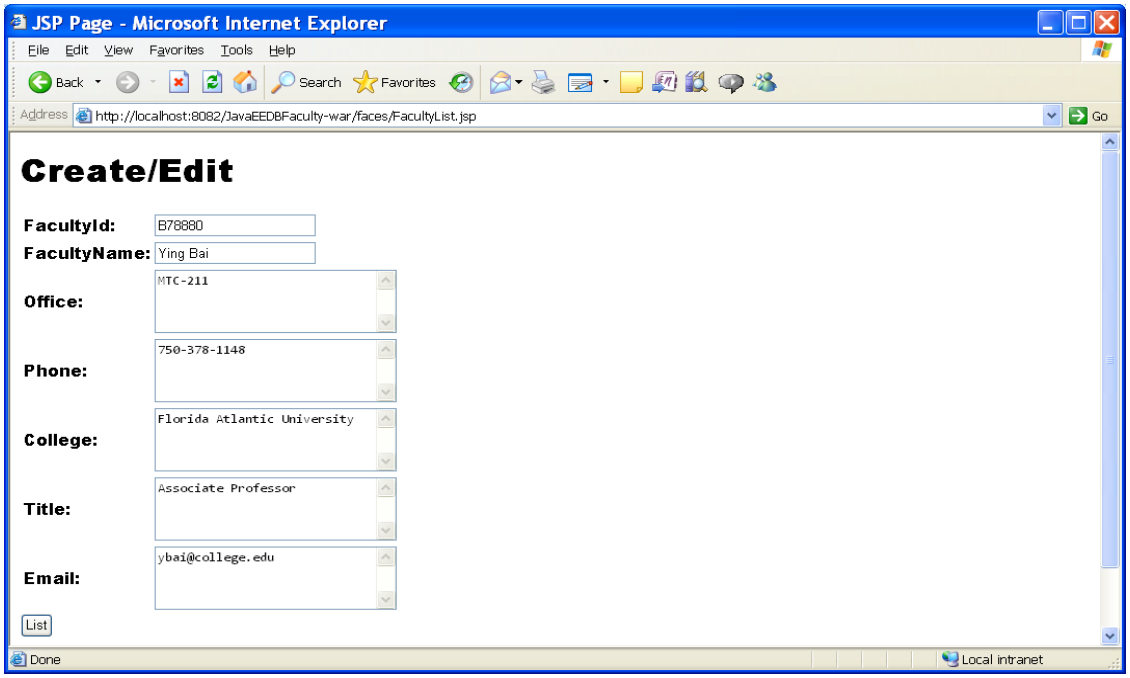

Figure K.34 A running result of the project JavaEEDBFaculty.