EJB integration in a struts application

Introduction

The usage of enterprise java beans in combination with struts is the goal of this tutorial. We create an EJB project and use the business processes (methods) later in the Struts application. As an example we use a library application, where you can manage books and users. In additional it should be possible, that a user can borrow a book.

General

Author:
Sascha Wolski
Sebastian Hennebrüder


Date:
November, 22nd 2004

Development Tools

Eclipse 3.x
MyEclipse plugin 3.8

Database
PostgreSQL 8.0 Beta

Application Server
Jboss 3.2.5

Source code:

http://www.laliluna.de/assets/tutorials/struts-ejb-integration.zip

The sources does not contain the project files. So create a project first and copy the sources to this projects.

PDF Version des Tutorials:

http://www.laliluna.de/assets/tutorials/struts-ejb-integration-en.pdf

Requirements

We are not repeating all the basics here. Have a look at the basic EJB tutorials before you start this tutorial.

Table of Contents

EJB integration in a struts application...........................................................................................................1
Introduction.........................................................................................................................................................1
General.................................................................................................................................................................1
Requirements.......................................................................................................................................................1
Overview............................................................................................................................................................2
Create a ejb project...........................................................................................................................................3
Create a database and a datasource file.............................................................................................................4
Configure xDoclet............................................................................................................................................4
Create the entity beans.....................................................................................................................................6
Edit the entity beans..........................................................................................................................................8
Add the relation................................................................................................................................................12
Create the session beans.................................................................................................................................15
Create the struts project..................................................................................................................................23
Create a default, welcome page......................................................................................................................26
Global Action Forwards and Action Mappings..............................................................................................28
Book list...........................................................................................................................................................30
Create a ejb project

Create a new enterprise java bean project (File > New > J2EE > EJB Project).

Set a nice name for the project. It is recommended that you add Ejb to the end of the project name, so it is evident that we are dealing with an ejb project.
Add a package for the entity and session beans

You have to add .ejb to the end of the package name. In the default configuration xDoclet expects EJBs to be placed in such packages.

Create a database and a datasource file

Create a new postgres database named ejbexample. Add a datasource file named ejbexample-ds.xml to the folder Jboss/server/deploy.

Examples for all kinds of supported databases can be found in the directory:

\jboss-3.2.5\docs\examples\jca

Add the following configuration to the file:

```xml
<datasources>
  <local-tx-datasource>
    <jndi-name>ejbexample</jndi-name>
    <connection-url>jdbc:postgresql://localhost:5432/ejbexample</connection-url>
    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>postgres</user-name>
    <password>pgsql</password>
  </local-tx-datasource>
</datasources>
```

Be sure that the port, the user and the password are correct for your database.

Configure xDoclet

Open the project properties (Strg + Enter). On „MyEclipse- Xdoclet“ right click in the upper right window and choose „Add Standard“, than „Standard EJB“. Now you will find „Standard EJB“ in the upper right window.
Choose "Standard EJB" and call the menu "Add" (right click) on the window below. Choose "jboss" from the list.

Now you find a menu entry jboss.

Set up xDoclet as shown in the picture below.
Thats all we need to do to configure xDoclet.

Create the entity beans

Create a new entity bean. Right click on the package > New > EJB(session...).

Set the properties like the picture below.
Create a second entity bean named User. Set it up like the bean before.
Edit the entity beans

First we have to edit the xDoclet configuration for the class.

**Entity Bean Book**

```java
/**
 * @author laliluna.de
 * @ejb.bean name = "Book"
 *     type = "CMP"
 *     cmp-version = "2.x"
 *     display-name = "Book"
 *     description = "Book EJB"
 *     view-type = "both"
 *     jndi-name = "ejb/BookHome"
 *     local-jndi-name = "ejb/BookLocalHome"
 *     primkey-field = "id"
 *     @ejb.util generate="physical"
 *     @ejb.persistence table-name = "tbooks"
 *     @ejb.value-object match = "*" name="Book"
 *     @ejb.finder signature = "java.util.Collection findAll()"
 *     query = "select object(c) from Book as c"
 *     @jboss.persistence create-table = "true"
 *     remove-table = "true"
 *     @jboss.entity-command name="postgresql-fetch-seq"
 */
public abstract class Book implements EntityBean {
  primkey-field = "id"
  @ejb.util generate="physical"
  Define the primary key field of the class
  @ejb.persistence table-name = "tbooks"
  Define how the Util class will generated and which lookup method are used. We use the physical
  JNDI lookup.
  @ejb.value-object match = "*" name="Book"
  Define the value object
  @ejb.finder signature = "java.util.Collection findAll()"
  Define a finder method
  query = "select object(c) from Book as c"
  @jboss.persistence create-table = "true"
  Define if the table will be automatically created on deploy and removed on undeploy.
  remove-table = "true"
  @jboss.entity-command name="postgresql-fetch-seq"
  Define which entity command is used for generating the bean.
}
```

Value Object Design Pattern

When you call a setName, setTitle method you create a roundtrip over the network between
Application Server and Database Server. Roundtrips costs time. This is why, people invented the
Value Object. Instead of working field per field a complete class instance is passed to methods or
returned by them.

In the next step we define the properties of the entity beans. Which fields are included on the
table.
The bean book has the following properties

- Id
- title
- author
- available
- user_id

Add the getter- and setter-methods for the properties:

```java
/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fid"
 *     sql-type = "SERIAL"
 *     jdbc-type = "INTEGER"
 * @ejb.pk-field
 *
 * @return
 */
public abstract Integer getId();

/**
 * @ejb.interface-method view-type = "both"
 * @param id
 */
public abstract void setId(Integer id);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "ftitle"
 * @return
 */
public abstract String getTitle();

/**
 * @ejb.interface-method view-type = "both"
 * @param title
 */
public abstract void setTitle(String title);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fauthor"
 * @return
 */
public abstract String getAuthor();

/**
 * @ejb.interface-method view-type = "both"
 * @param author
 */
public abstract void setAuthor(String author);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "favailable"
 * @return
 */
public abstract Boolean getAvailable();

/**
 * @ejb.interface-method view-type = "both"
 * @param available
 */
public abstract void setAvailable(Boolean available);
After adding the getter- and setter-methods run xDoclet.

Notice: Right click on the project > MyEclipse > Run xDoclet

xDoclet generated the value object of the entity bean, so we can add a getter and setter for this object.

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

public abstract BookValue getBookValue();

/**
* @ejb.interface-method view-type = "both"
* @param bookValue
*/

public abstract void setBookValue(BookValue bookValue);

Entity Bean User

First we also edit the xDoclet configuration for the class

/**
* @author laliluna.de
* * @ejb.bean name = "User"
* + type = "CMP"
* + cmp-version = "2.x"
* + display-name = "User"
* + description = "User EJB"
* + view-type = "both"
* + jndi-name = "ejb/UserHome"
* + local-jndi-name = "ejb/UserLocalHome"
* + primkey-field = "id"
* * @ejb.util generate="physical"
* @ejb.persistence table-name = "tusers"
* @ejb.value-object match = "*" name="User"
* * @ejb.finder signature = "java.util.Collection findAll()"
* + query = "select object(c) from User as c"
* * @jboss.persistence create-table = "true"
* remove-table = "true"
* @jboss.entity-command name="postgresql-fetch-seq"
*/

public abstract class User implements EntityBean {

The entity bean User has also four properties

• id
• lastName
**name**  
**age**  
Add the getter- and setter methods.

```java
/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fid"
 *       sql-type = "SERIAL"
 *       jdbc-type = "INTEGER"
 * @ejb.pk-field
 * @return
 */
public abstract Integer getId();

/**
 * @ejb.interface-method view-type = "both"
 * @param id
 */
public abstract void setId(Integer id);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fname"
 * @return
 */
public abstract String getName();

/**
 * @ejb.interface-method view-type = "both"
 * @param name
 */
public abstract void setName(String name);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "flastname"
 * @return
 */
public abstract String getLastName();

/**
 * @ejb.interface-method view-type = "both"
 * @param lastname
 */
public abstract void setLastName(String lastname);

/**
 * @ejb.interface-method view-type = "both"
 * @ejb.persistence column-name = "fage"
 * @return
 */
public abstract Integer getAge();

/**
 * @ejb.interface-method view-type = "both"
 * @param age
 */
public abstract void setAge(Integer age);
```

Run xDoclet. After it has generated the Value Object class, you can implement the getters and setters for the value object.

```java
/**
 * @ejb.interface-method view-type = "both"
 * @return
 */
```
public abstract UserValue getUserValue();

/**
 * @ejb.interface-method view-type = "both"
 * @param userValue
 */
public abstract void setUserValue(UserValue userValue);

Add the relation
In your application a user can borrow a book, so we need a relation between the book bean and the user bean. A user can borrow many books, but one book can only be borrow by one user. Its a one to many relation between user and books.

Add the getter- and setter-methods for the relation. Make sure that the interface viewtype is local.

**Book.java**

```java
/**
 * @ejb.interface-method view-type = "local"
 * @ejb.relation name = "user_book"
 *     role-name = "book_user"
 * @jboss.relation related-pk-field = "id"
 *     fk-column = "fuser_id"
 *     fk-constraint = "true"
 * @return
 */
public abstract UserLocal getUser();

/**
 * @ejb.interface-method view-type = "local"
 * @param user
 */
public abstract void setUser(UserLocal user);
```

**User.java**

```java
/**
 * @ejb.interface-method view-type = "local"
 * @ejb.relation name = "user_book"
 *     role-name = "user_book"
 * @return Collection
 */
public abstract Collection getBooks();

/**
 * @ejb.interface-method view-type = "local"
 * @param books
 */
public abstract void setBooks(Collection books);
```

**The view classes for the value objects**

It is possible that you edit the value object class, for example add the new method. But the problem arises when you run xDoclet again. It will replace the class and your method will be removed. Create a class that contain the value object and delegate the method of the class is a good and safe way to solve this problem. We name this class view, because it can contain additional methods for displaying, for example formatted float value or sum of values.

We create a new package de.laliluna.tutorial.library.view und add two new classes BookView and UserView. Define the value object BookValue in the class BookView and the same for UserValue in UserView. Generate the getter- and setter-methods and delegate all methods of the objects.
Delegate pattern (design pattern)

If you want to add particular properties to a class, it is the best way to use composition. In our case we want to extend the class `BookValue` with the user, which has borrowed the book.

You create an outer class, which contains the class as locale variable. The class `BookView` behaves like a book, so we can define for each method of the class `BookValue` a delegate method.

With eclipse you can create easily delegate methods. In your source code click the right mouse button and choose the function „Source“ -> „Delegate Methods“. Now you let MyEclipse create the delegate methods for you.

We need an additional object that contains the information of the user, who borrows the book, in your `BookView` class. We define an instance of the class `UserView` and generate the getter- and setter-methods.

The class looks like the following:

```java
public class BookView {

    private BookValue bookValue = new BookValue();
    private UserView userView = new UserView();

    public BookValue getBookValue() {
        return bookValue;
    }

    public void setBookValue(BookValue bookValue) {
        this.bookValue = bookValue;
    }

    public UserView getUserView() {
        return userView;
    }

    public void setUserView(UserView userView) {
        this.userView = userView;
    }

    public boolean authorHasBeenSet() {
        return bookValue.authorHasBeenSet();
    }

    public boolean availableHasBeenSet() {
        return bookValue.availableHasBeenSet();
    }

    public boolean equals(Object arg0) {
        return bookValue.equals(arg0);
    }

    public String getAuthor() {
        return bookValue.getAuthor();
    }

    public Boolean getAvailable() {
        return bookValue.getAvailable();
    }

    public Integer getId() {
        return bookValue.getId();
    }

    public Integer getPrimaryKey() {
        return bookValue.getPrimaryKey();
    }

    public String getTitle() {
        return bookValue.getTitle();
    }

    public Integer getUserId() {
        return bookValue.getUserId();
    }

    public int hashCode() {
        return bookValue.hashCode();
    }

    public boolean idHasBeenSet() {
        return bookValue.idHasBeenSet();
    }
}
```
public boolean isIdentical(Object other) {
    return bookValue.isIdentical(other);
}
public void setAuthor(String author) {
    bookValue.setAuthor(author);
}
public void setAvailable(Boolean available) {
    bookValue.setAvailable(available);
}
public void setId(Integer id) {
    bookValue.setId(id);
}
public void setPrimaryKey(Integer pk) {
    bookValue.setPrimaryKey(pk);
}
public void setTitle(String title) {
    bookValue.setTitle(title);
}
public void setUserId(Integer userId) {
    bookValue.setUserId(userId);
}
public boolean titleHasBeenSet() {
    return bookValue.titleHasBeenSet();
}
public String toString() {
    return bookValue.toString();
}
public boolean userIdHasBeenSet() {
    return bookValue.userIdHasBeenSet();
}
}

public class UserView {
    private UserValue userValue = new UserValue();

    public UserValue getUserValue() {
        return userValue;
    }
    public void setUserValue(UserValue userValue) {
        this.userValue = userValue;
    }
    public boolean ageHasBeenSet() {
        return userValue.ageHasBeenSet();
    }
    public boolean equals(Object arg0) {
        return userValue.equals(arg0);
    }
    public Integer getAge() {
        return userValue.getAge();
    }
    public Integer getId() {
        return userValue.getId();
    }
    public String getLastName() {
        return userValue.getLastName();
    }
    public String getName() {
        return userValue.getName();
    }
    public Integer getPrimaryKey() {
        return userValue.getPrimaryKey();
    }
    public int hashCode() {
        return userValue.hashCode();
    }
    public boolean idHasBeenSet() {

Create the session beans

*Book Session Bean*

Create a new session bean BookSession.
Right click on the package > New > EJB (Session...)
We want to add the following business processes to the session bean class `BookSession`.

- Get all existing books
- Get a book by primary key
- User borrow a book
- User return a book
- Update / add a book
- Remove a book

### Get all existing books

Within the method `getAllBooks()` we call a finder, that get all existing books. A Finder is a search method of the entity bean, which can return one or more beans.

We only want to use view object, so the method loops over the returned collection of the finder and fill an array with the view objects.

```java
/**
 * This method return all books as a collection
 * @ejb.interface-method view-type = "both"
 */
public BookView[] getAllBooks() throws EJBException {
    try {
        InitialContext context = new InitialContext();
        //Get the home interface with JNDI from the application server
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup
            (BookLocalHome.JNDI_NAME);
        //get all books with the local home interface
        Collection collection = bookLocalHome.findAll();
    }
```
Get a book by primary key

The method `getBookByPrimaryKey()` gets a book by the primary key and returns a `BookView` object.

```java
/** * This method gets a book by primary key * * @ejb.interface-method view-type = "both" */
public BookView getBookByPrimaryKey(Integer primaryKey) throws EJBException {
    try {
        InitialContext context = new InitialContext();
        BookView bookView = new BookView();

        // Get the home interface with JNDI from the application server
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);

        // Search for a book with the primary key
        BookLocal bookLocal = bookLocalHome.findByPrimaryKey(primaryKey);

        // Set the BookValue object in BookView
        bookView.setBookValue(bookLocal.getBookValue());

        // Set the UserValue in BookView
        if(bookLocal.getUser() != null)

        // Return the book value object
        return bookView;
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
```
User borrow a book

The method borrowBook() is called with the primary key of a book and a user. It saves the relation between the book and the user.

```java
/**
 * This method borrow a book for a user
 * @ejb.interface-method view-type = "both"
 */
public void borrowBook(Integer primaryKey, Integer userPrimaryKey) throws EJBException {
    try {
        InitialContext context = new InitialContext();
        // Get the home interface with JNDI from the application server
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);
        UserLocalHome userLocalHome = (UserLocalHome)context.lookup(UserLocalHome.JNDI_NAME);

        // find the book by primary key
        BookLocal bookLocal = bookLocalHome.findByPrimaryKey(primaryKey);

        // find the user by primary key
        UserLocal userLocal = userLocalHome.findByPrimaryKey(userPrimaryKey);

        // set the local interface of user to assign him to a book
        bookLocal.setUser(userLocal);
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
```

User return a book

This method remove the relation between one book and one user. The user return the book.

```java
/**
 * This method return a book from a user by primary key
 * @ejb.interface-method view-type = "both"
 */
public void returnBook(Integer primaryKey) throws EJBException {
    try {
        InitialContext context = new InitialContext();

        // Get the home interface with JNDI from the application server
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);

        // find a book by primary key
        BookLocal bookLocal = bookLocalHome.findByPrimaryKey(primaryKey);

        // remove the relation between the user
        bookLocal.setUser(null);
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
Update / add a book

The task of the method is saving a book. If the method cannot find the primary key it add a new book, otherwise it update the properties.

```java
/**
 * This method save the value object of the book
 */
public void saveBook(BookValue bookValue) throws EJBException {
    try {
        InitialContext context = new InitialContext();
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);
        //check if the book can be found for update, otherwise insert it as a new book
        BookLocal bookLocal = null;
        try {
            if(bookValue.getId()==null)bookValue.setId(new Integer(0));
            bookLocal = bookLocalHome.findByPrimaryKey(bookValue.getId());
        } catch (FinderException e1) {
            bookLocal = bookLocalHome.create();
        }
        //update the values of the book
        bookLocal.setBookValue(bookValue);
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (CreateException e) {
        throw new EJBException(e.getMessage());
    }
}
```

Remove a book

The method `removeBookByPrimaryKey()` deletes a book by its primary key.

```java
/**
 * This method removes a book from the database
 */
public void removeBookByPrimaryKey(Integer primaryKey) throws EJBException {
    try {
        BookLocalHome bookLocalHome = (BookLocalHome)context.lookup(BookLocalHome.JNDI_NAME);
        //find the book by primary key
        BookLocal bookLocal = bookLocalHome.findByPrimaryKey(primaryKey);
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (CreateException e) {
        throw new EJBException(e.getMessage());
    }
```
User Session Bean

Create a new session bean UserSession that contains the business processes of the users.

We also need methods for the following business methods.

- Get all users
- Get a user by primary key
- Update / add a user
- Delete a user

Get all users

This method gets all users and returns a collection.

```java
/**
 * This method returns all users as a collection
 */
```
Get a user by primary key

This method calls a finder to get user by primary key.

```java
/**
 * This method get a user by primary key
 * @ejb.interface-method view-type = "both"
 */
public UserValue getUserByPrimaryKey(Integer primaryKey) throws EJBException {
    try {
        InitialContext context = new InitialContext();
        // Get the home interface with JNDI from the application server
        UserLocalHome userLocalHome = (UserLocalHome) context.lookup(UserLocalHome.JNDI_NAME);
        // Find a user by primary key
        UserLocal userLocal = userLocalHome.findByPrimaryKey(primaryKey);
        // Return the book value object
        return userLocal.getUserValue();
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
```

Update / add a user

The task of this method is saving a user. If the primary key of the user cannot be found, a new user will be created, otherwise the properties of the user will be updated.

```java
/**
 * This method update or add a user.
 * @ejb.interface-method view-type = "both"
 */
public void saveUser(UserValue userValue) throws EJBException {
    try {
        InitialContext context = new InitialContext();
        // Get the home interface with JNDI from the application server
        UserLocalHome userLocalHome = (UserLocalHome) context.lookup(UserLocalHome.JNDI_NAME);
        // Find a user by primary key
        UserLocal userLocal = userLocalHome.findByPrimaryKey(primaryKey);
        // Return the book value object
        return userLocal.getUserValue();
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
```
//check if the user can be found for update, otherwise insert it as a new user
UserLocal userLocal = null;
try {
    if (userValue.getId() == null) userValue.setId(new Integer(0));
    userLocal = userLocalHome.findByPrimaryKey(userValue.getId());
} catch (FinderException e1) {
    userLocal = userLocalHome.create();
}

//set the value object of the local interface
userLocal.setUserValue(userValue);

} catch (NamingException e) {
    throw new EJBException(e.getMessage());
} catch (CreateException e) {
    throw new EJBException(e.getMessage());
}

---

**Remove a user**

This method removes a user by its primary key.

```java
/**
 * This method remove a user by primary key
 * @ejb.interface-method view-type = "both"
 */
public void removeUserByPrimaryKey(Integer primaryKey) throws EJBException {
    try {
        InitialContext context = new InitialContext();

        //Get the home interface with JNDI from the application server
        UserLocalHome userLocalHome = (UserLocalHome) context.lookup
        (UserLocalHome.JNDI_NAME);

        //find the user by primary key
        UserLocal userLocal = userLocalHome.findByPrimaryKey(primaryKey);

        //remove the user
        userLocal.remove();
    } catch (RemoveException e) {
        throw new EJBException(e.getMessage());
    } catch (NamingException e) {
        throw new EJBException(e.getMessage());
    } catch (FinderException e) {
        throw new EJBException(e.getMessage());
    }
}
```

**Congratulation, that's all.**
Create the dialogs

Create a new struts project with **File > New > Project** or use the shortcut **Strg + n**. Select the Wizard in J2EE **Web Project**.

Set a nice name for your project.
After creating the project, your Package Explorer looks like the picture below.
For now your project is still a normal J2EE project, so we need to add the Struts capabilities. Right click on the project and add the capabilities for Struts with Add Struts Capabilities.

Change the Base package for new classes and Default application resource

Configure the java build path
Open the project properties and select on java build path the ejb project „LibraryWebEjb“.
Create a default, welcome page

Ok, now we want to create a default page. Right click (yes again) on the Folder WebRoot in the Project and choose New > JSP.

Set the name to index.jsp and choose on template to use > Standard JSP using Struts 1.1 MyEclipse will use the template to create the JSP File.
You will find the file `index.jsp` in the folder `WebRoot` of the project. On the top of the file you will find the declaration of the struts tag libraries. These includes will be use to access the tags of struts. In this case we only need the logic tag library.

Insert the following line below the included logic tag.

```
<logic:forward name="welcome" />
```

This line instructs struts to look for a forward with the name `welcome`. If the application don’t find this forward it will state an error. In the next section I briefly explain the action forward.

Create a second `index.jsp` file in the folder `/WebRoot/jsp`
Global Action Forwards and Action Mappings

What is an action forward?
A action forward can be used to forward to a jsp or action mapping. There are two different action forwards. The global action forward and the local action forward. You can access a global action forward on each jsp or action class. A local action forward can only be accessed by the assigned action class.

What is a action mapping?
The action mapping is the heart of struts. It managed all actions between the application and the user. You can define which action will be executed by creating a action mapping.

The diagram show you, how the application server manage the request of the index.jsp or a non existing action mapping.

In the first step create a new action mapping. MyEclipse provides some nice features for creating struts files. Open the struts-config.xml, you will find it in the folder WebRoot/WEB-INF.

Click with the right mouse button on the entry action-mappings to create a new action with the wizard.
Choose **Use Case default** and **Action Type Forward**. The **Forward Path** is the welcome page `/jsp/index.jsp`.

To catch all requests of non-existing action mappings, we have to add manually a parameter `unknown="true"` to the action forward.

```xml
<action-mappings>
  <action forward="/jsp/index.jsp" path="/default" unknown="true"/>
</action-mappings>
```

Create the JSP specified above and change the code to the following:

```jsp
<%@ page language="java" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html locale="true">
<head>
<title>index.jsp</title>
<html:base />
<meta http-equiv="pragma" content="no-cache"/>
<meta http-equiv="cache-control" content="no-cache"/>
<meta http-equiv="expires" content="0"/>
<meta http-equiv="keywords" content="keyword1,keyword2,keyword3"/>
<meta http-equiv="description" content="This is my page"/>
</head>
<body>
```
Welcome!

In the second step you create a global action forward. Go back to the outline window of MyEclipse and choose **Global Forward**.

Choose the **Forward Scope** **Global Forward**. For name use the same you have set in your default page. The **Global Forward** refers to your action mapping.

You will see the following in your editor window.

```xml
<global-forwards>
  <forward name="welcome" path="/default.do" redirect="true" />
</global-forwards>
<action-mappings>
  <action forward="/jsp/index.jsp" path="/default" unknown="true" />
</action-mappings>
```
Book list

This use case list all available books.

Open the struts-config.xml. Right click on Form Bean in the outline window.

Use Case is bookList, Superclass org.apache.struts.action.ActionForm. Select public void reset.. to create this method. Set the name of the jsp file on the JSP tab.
The package explorer looks like the pictures below.

---

**Edit the source code of the action form class**

Open the file `BookListForm.java` and add the following.

```java
public class BookListForm extends ActionForm {
    private BookView[] bookViews = new BookView[0];
    public BookView[] getBookViews() {
        return bookViews;
    }
    public void setBookViews(BookView[] bookViews) {
        this.bookViews = bookViews;
    }
}
```

Define an array of type `BookView` and generate a getter and setter.

---

**Action mapping and action class of the book list**

Open the `struts-config.xml` and create a new action mapping.

Use Case is bookList, choose Create new Action Class
Superclass `org.apache.struts.Action`
On Optional Details choose the Form Bean `bookListForm`
The input source is `/jsp/bookList.jsp`
Now add a forward `showList` to the action mapping.

You will find the action class `bookListAction` in your package `de.laliluna.tutorial.library.action`. 
**Edit the source code of the action class**

Open the class **bookListAction** and edit the method **execute**. Create a local home interface of the class **BookSession** and call the method that gets all books. Save the array of books returned by the method in the form bean. The command **mapping.findForward** ("showList") will search for a local forward with the name **showList**.

```java
/**
 * Method execute
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    BookListForm bookListForm = (BookListForm) form;
    try {
        InitialContext context = new InitialContext();
        //holen des Home Interfaces für BookSession mit JNDI vom Appplication Server
        BookSessionHome bookSessionHome = (BookSessionHome) context.lookup
                                      (BookSessionHome.JNDI_NAME);
        BookSession bookSession = bookSessionHome.create();
        //liest alle Bücher aus und setze diese im Form Bean
        bookListForm.setBookViews(bookSession.getAllBooks());
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showList");
}
```

Yeah that's all, you have now created your form bean with an action form class, an action mapping with an action class and the jsp to display something.

**Display the books list in the jsp file.**

Open the **bookList.jsp** and add the following source code.

```html
<%@ page language="java" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>

<html>
    <head>
        <title>Show book list</title>
    </head>
    <body>
```
```html
<table border="1">
  <tbody>
    <!-- set the header -->
    <tr>
      <td>Author</td>
      <td>Book name</td>
      <td>Available</td>
      <td>Borrowed by</td>
    </tr>
    <!-- start with an iterate over the array bookViews -->
    <logic:iterate name="bookListForm" property="bookViews" id="book">
      <!-- book informations -->
      <td><bean:write name="book" property="author" /></td>
      <td><bean:write name="book" property="title" /></td>
      <td><html:checkbox disabled="true" name="book" property="available" /></td>
      <td><logic:notEmpty name="book" property="userView.name">
          <bean:write name="book" property="userView.name" />, 
          <bean:write name="book" property="userView.lastName" />
        </logic:notEmpty>
      </td>
    </logic:iterate>
    <!-- borrow, edit and delete link for each book -->
    <td>
      <!-- check if a user borrowed a book, when its true display the return link otherwise display nothing -->
      <logic:notEmpty name="book" property="userView.name">
        <html:link action="bookEdit.do?do=returnBook"
          paramName="book"
          paramProperty="id"
          paramId="id">Return <book> </html:link>
      </logic:notEmpty>
      <logic:empty name="book" property="userView.name">
        <html:link action="bookEdit.do?do=borrowBook"
          paramName="book"
          paramProperty="id"
          paramId="id">Borrow <book> </html:link>
      </logic:empty>
    </td>
    <td>
        paramName="book"
        paramProperty="id"
        paramId="id">Edit </html:link>
    </td>
    <td>
      <html:link action="bookEdit.do?do=deleteBook"
        paramName="book"
        paramProperty="id"
        paramId="id">Delete </html:link>
    </td>
  </logic:iterate>
  <!-- end iterate -->
  <!-- if books cannot be found display a text -->
  <logic:iterate>
    <logic:empty name="book" property="userView.name">
      <html:link action="bookEdit.do?do=borrowBook"
        paramName="book"
        paramProperty="id"
        paramId="id">Borrow <book> </html:link>
    </logic:empty>
    <logic:notEmpty name="book" property="userView.name">
      <html:link action="bookEdit.do?do=returnBook"
        paramName="book"
        paramProperty="id"
        paramId="id">Return <book> </html:link>
    </logic:notEmpty>
  </logic:iterate>
</tbody>
</table>
```
The tag `<logic:iterate>` loops over the array of books. Within the tag you have access to the properties of the books with the name book. The tag `<bean:write>` prints out a property of a book, for example the title. With the tag `<logic:notEmpty>` and `<logic:empty>` we check, if a user has borrowed a book or not.

**Add, edit, borrow and delete books**

In the next step we have to add the following use cases.
- Add books
- Edit books
- Borrow / return books
- Delete books

**New form bean**

Create a new form bean and action form class. Set Use case to bookEdit and remove all methods on Optional details - Methods. Let MyEclipse create the jsp file for us.

Open the class `BookEditForm.java` in `de.laliluna.tutorial.library.form`.

Create a new instance `bookView` of the class `BookView`

```java
private BookView bookView = new BookView();
```

Generate a getter and setter for this instance.
Delegate all methods of the class BookView.

```java
public class BookEditForm extends ActionForm {

    private BookView bookView = new BookView();

    public BookView getBookView() {
        return bookView;
    }

    public void setBookView(BookView bookView) {
        this.bookView = bookView;
    }

    public boolean authorHasBeenSet() {
        return bookView.authorHasBeenSet();
    }

    public boolean availableHasBeenSet() {
        return bookView.availableHasBeenSet();
    }

    public boolean equals(Object arg0) {
        return bookView.equals(arg0);
    }

    public String getAuthor() {
        return bookView.getAuthor();
    }

    public Boolean getAvailable() {
        return bookView.getAvailable();
    }

    public BookValue getBookValue() {
        return bookView.getBookValue();
    }

    public Integer getId() {
```

public Integer getPrimaryKey() {
    return bookView.getPrimaryKey();
}

public String getTitle() {
    return bookView.getTitle();
}

public Integer getUserId() {
    return bookView.getUserId();
}

public UserView getUserView() {
    return bookView.getUserView();
}

public Integer getId() {
    return bookView.getId();
}

public Integer getPrimaryKey() {
    return bookView.getPrimaryKey();
}

public String getTitle() {
    return bookView.getTitle();
}

public Integer getUserId() {
    return bookView.getUserId();
}

public UserView getUserView() {
    return bookView.getUserView();
}

public int hashCode() {
    return bookView.hashCode();
}

public boolean idHasBeenSet() {
    return bookView.idHasBeenSet();
}

public boolean isIdentical(Object other) {
    return bookView.isIdentical(other);
}

public void setAuthor(String author) {
    bookView.setAuthor(author);
}

public void setAvailable(Boolean available) {
    bookView.setAvailable(available);
}

public void setBookValue(BookValue bookValue) {
    bookView.setBookValue(bookValue);
}

public void setId(Integer id) {
    bookView.setId(id);
}

public void setPrimaryKey(Integer pk) {
    bookView.setPrimaryKey(pk);
}

public void setTitle(String title) {
    bookView.setTitle(title);
}

public void setUserId(Integer userId) {
    bookView.setUserId(userId);
}

public void setUserView(UserView userView) {
    bookView.setUserView(userView);
}

public boolean titleHasBeenSet() {
    return bookView.titleHasBeenSet();
}

public String toString() {
    return bookView.toString();
}

public boolean userIdHasBeenSet() {
    return bookView.userIdHasBeenSet();
}
**Action Mapping**

Create a new action mapping. There is a different between our first action class. The new action class will extends to the superclass `org.apache.struts.DispatchAction`.

On Parameter we add a parameter `do`. These parameter is needed by the dispatch action class.

Add four new forwards. One is for the edit page, the second for the add page, where you can add the books, the third forward to the borrow page and the last forward redirect the user to the book listing.
The last forward is different to the others. It refers to an existing action mapping and redirect the user.

Create the non existing jsp files with New > JSP.

bookAdd.jsp
bookBorrow.jsp
**Create the source code of the jsp files**

Open the file `bookAdd.jsp` and add the following source code.

```html
<%@ page language="java"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic"%>

<html>
<head>
    <title>Add a book</title>
</head>
<body>
    <!-- create a html form -->
    <html:form action="bookEdit">
    <!-- print out the form data -->
    <table border="1">
      <tr>
        <td>Author:</td>
        <td><html:text property="author" /></td>
      </tr>
      <tr>
        <td>Title:</td>
        <td><html:text property="title" /></td>
      </tr>
      <tr>
        <td>Available:</td>
        <td><html:checkbox property="available" /></td>
      </tr>
    </table>
    <!-- set the parameter for the dispatch action -->
    <html:hidden property="do" value="saveBook" />
    </html:form>
</body>
</html>
```

The tag `<html:form>` creates a new HTML form and refers with the parameter `action="bookEdit"` to the action mapping. The Tag `<html:text>` creates a text field with the property `author` of the book. `<html:hidden>` is a hidden form field with the name `do`. We need this hidden field, because it tells the dispatch action class which method will called.

Open the file `bookEdit.jsp`. You can use the source code of the of the file `bookAdd.jsp` and change the following lines.

```html
<title>Edit a book</title>
```

Add the following line above `<html:hidden property="do" value="saveBook" />

```html
<!-- hidden fields for id and userId -->
<html:hidden property="id" />
<html:hidden property="userId" />
```
Open the file bookBorrow.jsp and add the following.

```html
<%@ page language="java"%>
<%@ page isELIgnored="false"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic"%>

<html>
<head>
<title>Show user</title>
</head>
<body>
<html:form action="bookEdit">
<table border="1">
<tr>
    <td>Last name</td>
    <td>Name</td>
    <td>Borrowed</td>
</tr>
<logic:iterate name="users" id="user">
    <tr>
        <td><bean:write name="user" property="lastName" /></td>
        <td><bean:write name="user" property="name" /></td>
        <td><html:radio property="userId" value="${user.id}" /></td>
    </tr>
</logic:iterate>
</table>
<logic:notPresent name="user">
    <tr>
        <td colspan="5">No users found.</td>
    </tr>
</logic:notPresent>
</html:form>
</body>
</html>
Ok, thats all.
Methods of the dispatch action class

DispatchAction

Normally you set up each action event to always correspond to a unique Action class. There is nothing wrong with doing this but if you want to condense related events into one Action class you can do this by using a DispatchAction class or some of its sub classes.

Open the file `bookEditAction.java` and add the following methods

```java
public ActionForward editBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {
    BookEditForm bookEditForm = (BookEditForm) form;

    /* lalinuna.de 04.11.2004
    * get id of the book from request
    */
    Integer id = Integer.valueOf(request.getParameter("id"));

    /* lalinuna.de 16.11.2004
    * load the session facade and get the book by primary key
    */
    try {
        InitialContext context = new InitialContext();
        //holen des Home Interfaces für BookSession mit JNDI vom Appplication Server
        BookSessionHome bookSessionHome = (BookSessionHome)context.lookup(BookSessionHome.JNDI_NAME);
        BookSession bookSession = bookSessionHome.create();
        //bestimme das Buch anhand seines Primärschlüssels und setzte es im Form Bean
        bookEditForm.setBookValue(bookSession.getBookByPrimaryKey(id));
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showEdit");
}

The method `editBook` gets the id from the request and gets the book by the primary key from the database. The forward `showEdit` refers to the jsp file where you can edit the book.

public ActionForward borrowBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {
    BookEditForm bookEditForm = (BookEditForm) form;

    /* lalinuna.de 04.11.2004
    * get id of the book from request
    */
    Integer id = Integer.valueOf(request.getParameter("id"));

    /* lalinuna.de 16.11.2004
    * load the session facade for book and user
    * get the book information and get all users
    */
    try {
        InitialContext context = new InitialContext();
```
```
//holen des Home Interfaces für BookSession mit JNDI vom Application Server
BookSessionHome bookSessionHome = (BookSessionHome)context.lookup(BookSessionHome.JNDI_NAME);
BookSession bookSession = bookSessionHome.create();

//holen des Home Interfaces für UserSession mit JNDI vom Application Server
UserSessionHome userSessionHome = (UserSessionHome)context.lookup(UserSessionHome.JNDI_NAME);
UserSession userSession = userSessionHome.create();

//liest das Buch anhand seines Primärschlüssels und setzte es im Form Bean
bookEditForm.setBookView(bookSession.getBookByPrimaryKey(id));

//bestimme alle Benutzer und setzte diese im Request
request.setAttribute("users", userSession.getAllUsers());

} catch (RemoteException e) {
    e.printStackTrace();
} catch (NamingException e) {
    e.printStackTrace();
} catch (CreateException e) {
    e.printStackTrace();
}

return mapping.findForward("showBorrowUser");

The method contains the process that a user can borrow a book. It assigns a user to a book.

public ActionForward returnBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {
    BookEditForm bookEditForm = (BookEditForm) form;

    /* lalinuna.de 04.11.2004 */
    Integer id = Integer.valueOf(request.getParameter("id"));

    /* lalinuna.de 16.11.2004 */
    try {
        InitialContext context = new InitialContext();

        //holen des Home Interfaces für BookSession mit JNDI vom Application Server
        BookSessionHome bookSessionHome = (BookSessionHome)context.lookup(BookSessionHome.JNDI_NAME);
        BookSession bookSession = bookSessionHome.create();

        //löscht die Beziehung zwischen Benutzer und Buch
        bookSession.returnBook(id);
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showList");
The next method removes the relation between a user and a book, the book is returned by a user.

```java
public ActionForward deleteBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    BookEditForm bookEditForm = (BookEditForm) form;

    /* lalinuna.de 04.11.2004 */
    Integer id = Integer.valueOf(request.getParameter("id"));

    /* lalinuna.de 16.11.2004 */
    try {
        InitialContext context = new InitialContext();
        BookSessionHome bookSessionHome = (BookSessionHome)context.lookup(BookSessionHome.JNDI_NAME);
        BookSession bookSession = bookSessionHome.create();

        bookSession.removeBookByPrimaryKey(id);
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showList");
}
```

The method `deleteBook` gets the parameter `id` of the request and deletes the book from the database. Then the forward `showList` displays the book list page.

```java
public ActionForward addBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    BookEditForm bookEditForm = (BookEditForm) form;

    return mapping.findForward("showAdd");
}
```

The method `addBook` refers to the `bookAdd.jsp`
This method assigns a user to a book, if a user borrows a book.

```java
public ActionForward saveBook(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {
    BookEditForm bookEditForm = (BookEditForm) form;

    try {
        InitialContext context = new InitialContext();
        BookSessionHome bookSessionHome = (BookSessionHome)context.lookup
            (BookSessionHome.JNDI_NAME);
        BookSession bookSession = bookSessionHome.create();

        bookSession.saveBook(bookEditForm.getBookValue());
    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showList");
}
```

The last method updates a book.

**User list**

We create this list on the same way like the book list.

Open the `struts-config.xml`. Right click on Form Bean in the outline window.

Use Case is `userList`, Superclass `org.apache.struts.ActionForm` On Methods choose only `public void reset`. On JSP set a name for the jsp file.
Edit the source code of the action class

Open the file UserListForm.java and add the following source code.

```java
public class UserListForm extends ActionForm {

    private Collection users;

    /* lalinuna.de 02.11.2004 */
    /* get the collection books */
    public Collection getUsers() {
        return users;
    }

    /* lalinuna.de 02.11.2004 */
    /* set the collection books */
    public void setUsers(Collection users) {
        this.users = users;
    }

    /* lalinuna.de 02.11.2004 */
    /* reset the collection books */
    public void reset(ActionMapping arg0, HttpServletRequest arg1) {
        users = new ArrayList();
    }
}
```

We define a collection `users` and generate a getter and setter. Within the method `reset` we initialize the collection as a new array list.
**Action mapping and action class for the user list**

Open the struts-config.xml and create a new action mapping.

**Use Case** userList, choose Create new Action Class

**Superclass** org.apache.struts.Action

Choose on Optional Details the form bean userListForm and on Input Source the jsp file /jsp/userList.jsp

---

**Edit the action class**

Open the file UserListAction.java and add the following source code.

```java
/**
 * Method execute
 * @param mapping
 */
```
The method gets all users and set them in the form bean.

**Displaying the book list in the jsp file**

Open the jsp file `userList.jsp` and change the content of the file like the following.

```html
<%@ page language="java"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic"%>

<html>
<head>
<title>Show user list</title>
</head>
<body>

<table border="1">
<tbody>
<%-- set the header --%>
<tr>
<td>Last name</td>
<td>Name</td>
<td>Age</td>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<%-- start with an iterate over the collection users --%>
<logic:iterate name="userListForm" property="users" id="user">
<tr>
```
<%-- book informations --%>
<td><bean:write name="user" property="lastName" /></td>
<td><bean:write name="user" property="name" /></td>
<td><bean:write name="user" property="age" /></td>

<%-- edit and delete link for each book --%>
<td><html:link action="userEdit.do?do=editUser"
    paramName="user"
    paramProperty="id"
    paramId="id">Edit</html:link></td>
<td><html:link action="userEdit.do?do=deleteUser"
    paramName="user"
    paramProperty="id"
    paramId="id">Delete</html:link></td>
</tr>

<logic:iterate>
<%-- end interate --%>
<%-- if users cannot be found display a text --%>
<logic:notPresent name="user">
    <tr>
        <td colspan="5">No users found.</td>
    </tr>
</logic:notPresent>
</tbody>
</table>
<br>
<%-- add and back to menu button --%>
<html:button property="add"
    onclick="location.href='userEdit.do?do=addUser'">Add a new user</html:button>
<html:button property="back"
    onclick="location.href='default.do'">Back to menu</html:button>
</html>
Add, edit, delete users

In the next step we want to add the following use cases.

- Add a user
- Edit a user
- Delete User

New form bean

Add a new form bean, with an action form class. Set the Use case userEdit. Choose on Optional details – Methods no methods. On JSP select that the jsp file will be created.

Open the class UserEditForm.java on de.laliluna.tutorial.library.form.
Add a new instance of userView of the class UserView.

```java
private UserView userView = new UserView();
```

Generate a getter- and setter-method and delegate all methods of the class, like you have done it with the book form bean.

```java
public class UserEditForm extends ActionForm {
    private UserView userView = new UserView();
    
    public UserView getUserView() {
        return userView;
    }
    
    public void setUserView(UserView userView) {
        this.userView = userView;
    }
    
    public boolean ageHasBeenSet() {
        return userView.ageHasBeenSet();
    }
}
```
public boolean equals(Object arg0) {
    return userView.equals(arg0);
}

public Integer getAge() {
    return userView.getAge();
}

public Integer getId() {
    return userView.getId();
}

public String getLastName() {
    return userView.getLastName();
}

public String getName() {
    return userView.getName();
}

public Integer getPrimaryKey() {
    return userView.getPrimaryKey();
}

public UserValue getUserValue() {
    return userView.getUserValue();
}

public int hashCode() {
    return userView.hashCode();
}

public boolean idHas BeenSet() {
    return userView.idHas BeenSet();
}

public boolean isIdentical(Object other) {
    return userView.isIdentical(other);
}

public boolean lastNameHas BeenSet() {
    return userView.lastNameHas BeenSet();
}

public boolean nameHas BeenSet() {
    return userView.nameHas BeenSet();
}

public void setAge(Integer age) {
    userView.setAge(age);
}

public void setId(Integer id) {
    userView.setId(id);
}

public void setLastName(String lastName) {
    userView.setLastName(lastName);
}

public void setName(String name) {
    userView.setName(name);
}

public void setPrimaryKey(Integer pk) {
    userView.setPrimaryKey(pk);
}

public void setUserValue(UserValue userValue) {
    userView.setUserValue(userValue);
}

public String toString() {
    return userView.toString();
}
Open the `struts-config.xml` and create a new action mapping.

Set the Use case to `userEdit`. Choose Create new Action class and on Superclass `org.apache.struts.actions.DispatchAction`. On Optional Details > Form choose the form bean.
Set the parameter on Option Details to do.

At the end add three forwards to the different pages.

**Edit the source code of the action class**

Open the class `UserEditAction.java` in the package `de.laliluna.tutorial.library.action` and add the following methods.

```java
/** *
 * Method editBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward editUser(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    UserEditForm userEditForm = (UserEditForm) form;

    try {
        InitialContext context = new InitialContext();
        //holen des Home Interfaces für UserSession mit JNDI vom Application Server
    }
```
UserSessionHome userSessionHome = (UserSessionHome) context.lookup (UserSessionHome.JNDI_NAME);
UserSession userSession = userSessionHome.create();

// bestimmen eines Benutzers anhand seines Primärschlüssels und setzen im Form Bean
userEditForm.setUserValue(userSession.getUserByPrimaryKey(id));

} catch (RemoteException e) {
e.printStackTrace();
} catch (NamingException e) {
e.printStackTrace();
} catch (CreateException e) {
e.printStackTrace();
}

return mapping.findForward("showEdit");

This method gets a user by his primary key and save them in the form bean.

/**
 * Method deleteBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward deleteUser(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    UserEditForm userEditForm = (UserEditForm) form;

    /* lalinuna.de 04.11.2004
    * get id of the book from request
    */
    Integer id = Integer.valueOf(request.getParameter("id"));

    /* lalinuna.de 16.11.2004
    * load the session facade and delete the user by primary key
    */
    try {
        try {
            InitialContext context = new InitialContext();
            // holen des Home Interfaces für UserSession mit JNDI vom Application_Server
            UserSessionHome userSessionHome = (UserSessionHome) context.lookup (UserSessionHome.JNDI_NAME);
            UserSession userSession = userSessionHome.create();
            // löschen eine Benutzer anhand seines Primärschlüssels
            userSession.removeUserByPrimaryKey(id);
        } catch (RemoteException e) {
            e.printStackTrace();
        } catch (NamingException e) {
            e.printStackTrace();
        } catch (CreateException e) {
            e.printStackTrace();
        }

        return mapping.findForward("showUserList");
    }

The next method deletes a user by primary key and forwards to the user list page.
/**
 * Method addUser
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward addUser(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    UserEditForm userEditForm = (UserEditForm) form;
    return mapping.findForward("showAdd");
}

This method forwards to the add dialog

/**
 * Method saveUser
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward saveUser(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) {

    UserEditForm userEditForm = (UserEditForm) form;

    /* lalinuna.de 16.11.2004
     * load the session facade and save the book by primary key
     */
    try {
        InitialContext context = new InitialContext();
        //holen des Home Interfaces für UserSession mit JNDI vom
        Appplication Server
        UserSessionHome userSessionHome = (UserSessionHome)context.lookup
            (UserSessionHome.JNDI_NAME);
        UserSession userSession = userSessionHome.create();

        //aktualisiert oder legt eine neues Buch an
        userSession.saveUser(userEditForm.getUserValue());

    } catch (RemoteException e) {
        e.printStackTrace();
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (CreateException e) {
        e.printStackTrace();
    }

    return mapping.findForward("showUserList");
}

This method updates and adds a user, like the method in the book action class. It forwards to the user list page.
**Edit the source code of the jsp files**

Create a new file named `userAdd.jsp` in the folder `WebRoot/jsp/`.

Open the file `userAdd.jsp` and change the content of the file.

```html
<%@ page language="java" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>

<html>
<head>
<title>Add a user</title>
</head>
<body>

<!-- create a html form -->
<html:form action="userEdit">

<!-- print out the form data -->
<table border="1">
<tbody>
<tr>
<td>Last name:</td>
<td><html:text property="lastName" /></td>
</tr>
<tr>
<td>Name:</td>
<td><html:text property="name" /></td>
</tr>
<tr>
<td>Age:</td>
<td><html:text property="age" /></td>
</tr>
</tbody>
</table>

<!-- set the parameter for the dispatch action -->
<html:hidden property="do" value="saveUser" />
<br/>

<!-- submit and back button -->
<html:button property="back" onclick="history.back();">
Back
</html:button>

&nbsp;
<html:submit>Save</html:submit>
</html:form>
</body>
</html>

It is quite the same like the book edit page.

Open the file `userEdit.jsp` copy the source code of the file `userAdd.jsp` and add above the line `<html:hidden property="do" value="saveUser" />` the following

```html
<!-- hidden field that contains the id of the user -->
<html:hidden property="id" />
```
Test the applications
Start the jboss and deploy the project as package archiv.

Call the project in your favorite web browser. http://localhost:8080/LibraryWeb/

Nice, that's all.