First steps in Struts using eclipse + MyEclipse

This tutorial will explain first steps using the web framework Apache Struts and the development environment eclipse. We will create a simple example library application.

Generals

Author:
Sascha Wolski

http://www.laliluna.de/tutorial.html – Tutorials for Struts, EJB, xdoclet and eclipse.

Date:
November, 2st 2004
Software:
MyEclipse plugin 3.8

(A cheap and quite powerful Extension to Eclipse to develop Web Applications and EJB (J2EE) Applications. I think that there is a test version availalable at MyEclipse.)

Source code:

http://www.laliluna.de/assets/tutorials/first_steps_with_struts.zip

Using the source code.

The source code does not include any libraries but the sources. Create a web project, add the libraries manually or with the help of MyEclipse and the extract the sources we provided to your project.

The PDF Version of the tutorial.

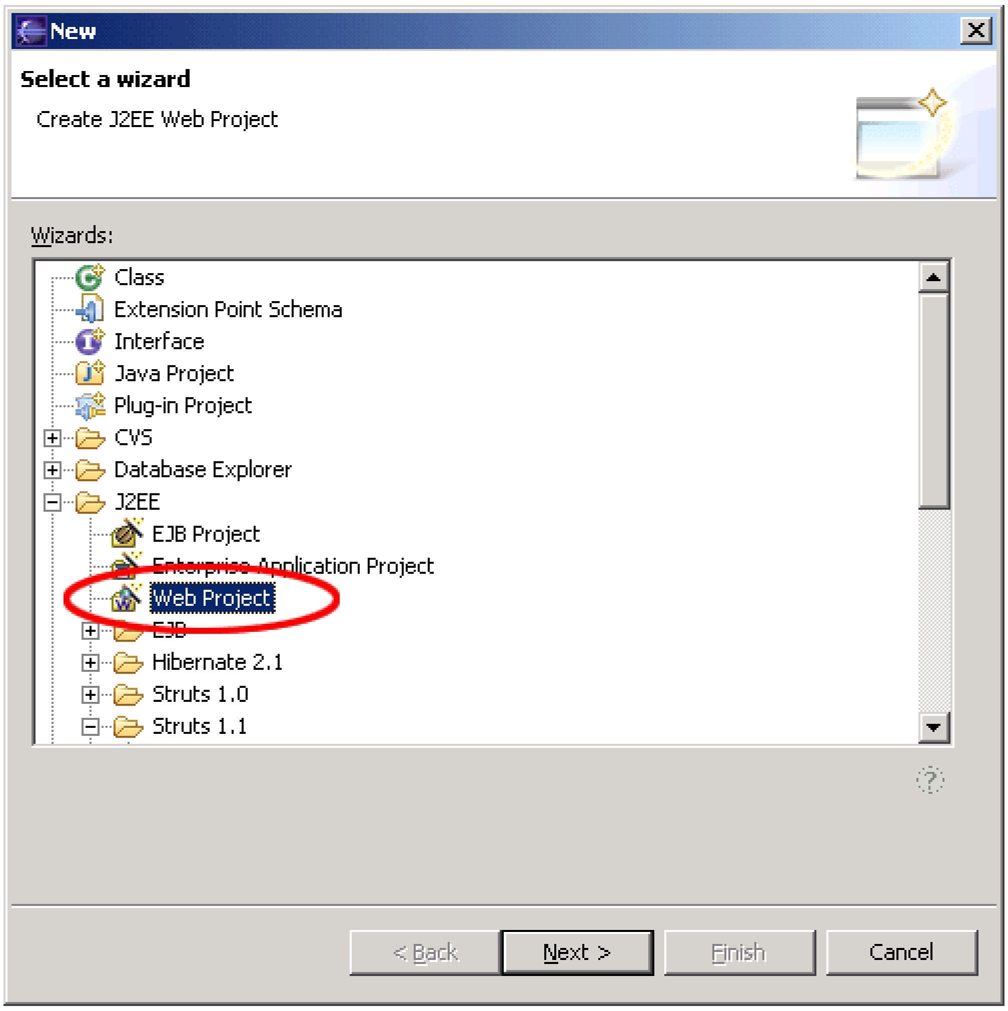
http://www.laliluna.de/assets/tutorials/first_steps_with_struts_en.pdf

Table of Content

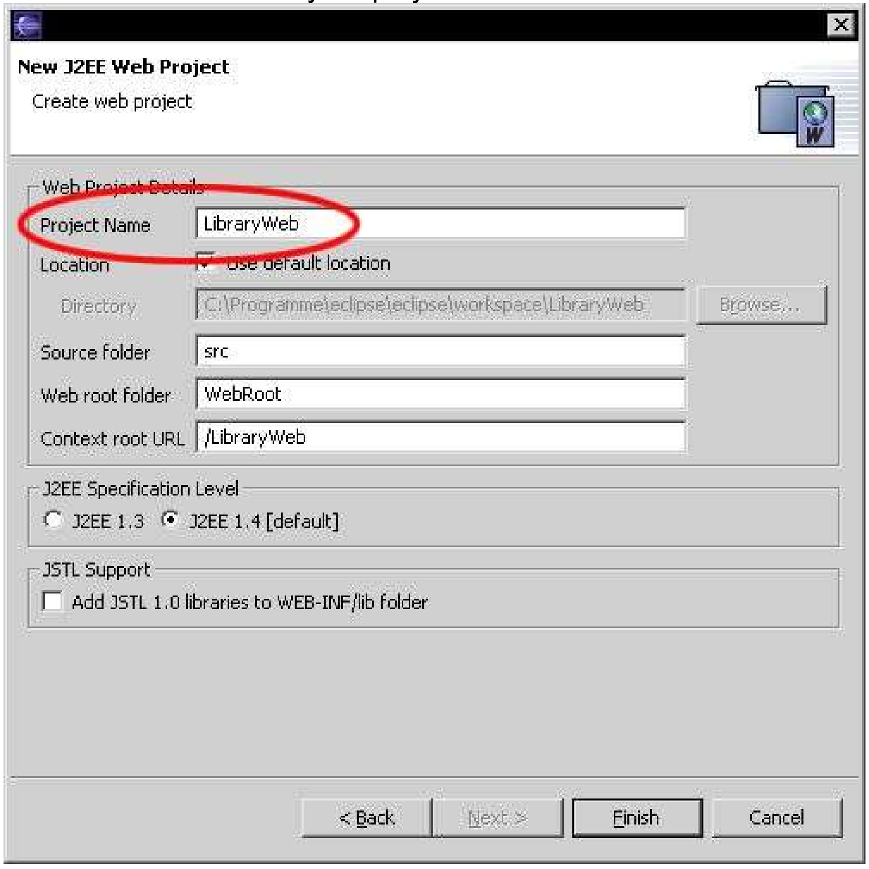
First steps in Struts using eclipse + MyEclipse.......... 1
Generals.......... 1
Create a struts project.......... 1
Global Action Forwards and Action Mappings.......... 6
Create a object class „book“.......... 9
Create a form bean, action form and jsp.......... 13
Edit the source code of the action form class.......... 14
Create an action mapping and action class.......... 15
Class to provide test data.......... 16
Edit the source code of the action class.......... 16
Output the test data on the jsp file.......... 17
Add, edit and remove the data.......... 18
Edit the book listing page.......... 24
Test the application.......... 25

Create a struts project

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



Create a nice name for your project


New J2EE Web Project

Create web project

Web Project Details

Project Name	LibraryWeb
Location	[✓] Use default location
Directory	C:\Programme\eclipse\eclipse\workspace\LibraryWeb
Source folder	src
Web root folder	WebRoot
Context root URL	/LibraryWeb

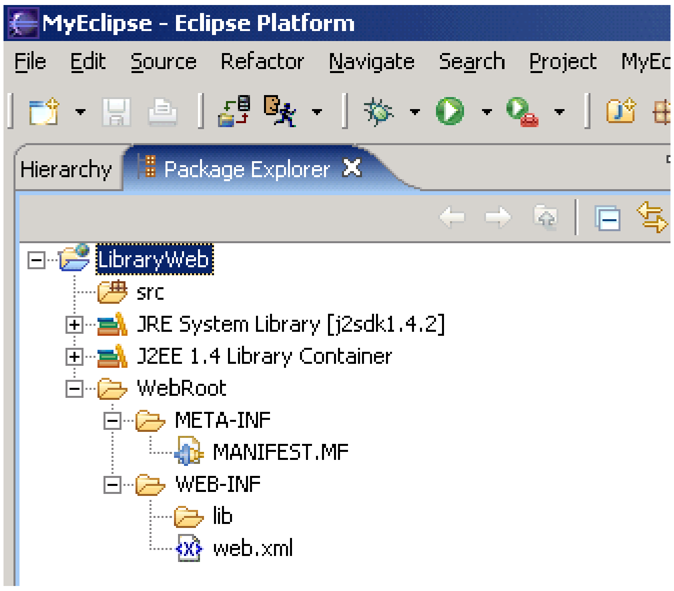
J2EE Specification Level

J2EE 1.3 [✓ J2EE 1.4 [default]]

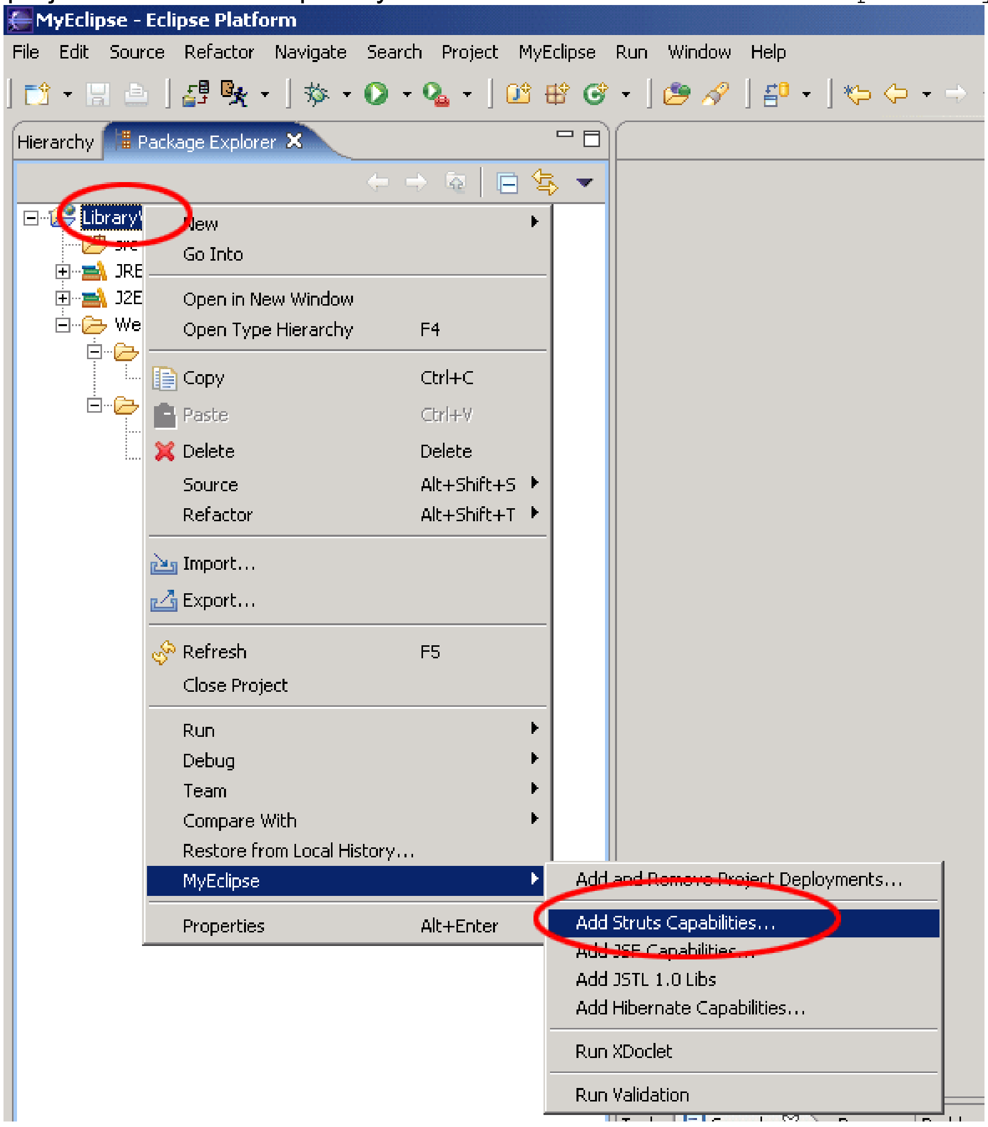
JSTL Support

[ ] Add JSTL 1.0 libraries to WEB-INF/lib folder

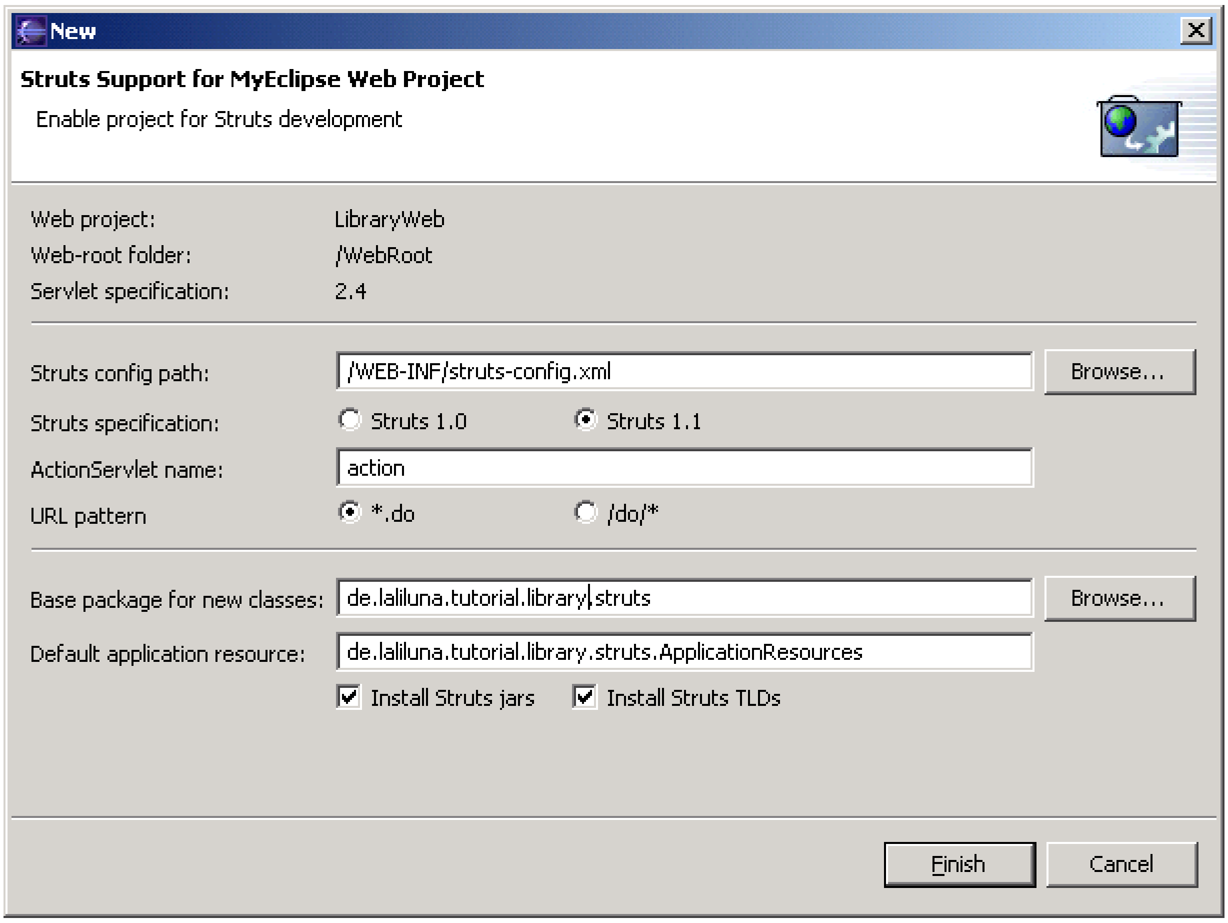
After creating the project, your Package Explorer looks like the picture below.


MyEclipse - Eclipse Platform
File Edit Source Refactor Navigate Search Project MyEc
Hierarchy
Package Explorer
LibraryWeb
src
JRE System Library [j2sdk1.4.2]
J2EE 1.4 Library Container
WebRoot
META-INF
MANIFEST.MF
WEB-INF
lib
web.xml

For now your project is a normal J2EE project, so we need to add the struts capabilityies. Right click on the project and add the capabilityies for struts with `Add Struts Capabilityies.`



Change the properties `Base package for new classes` and `Default application resource`



New

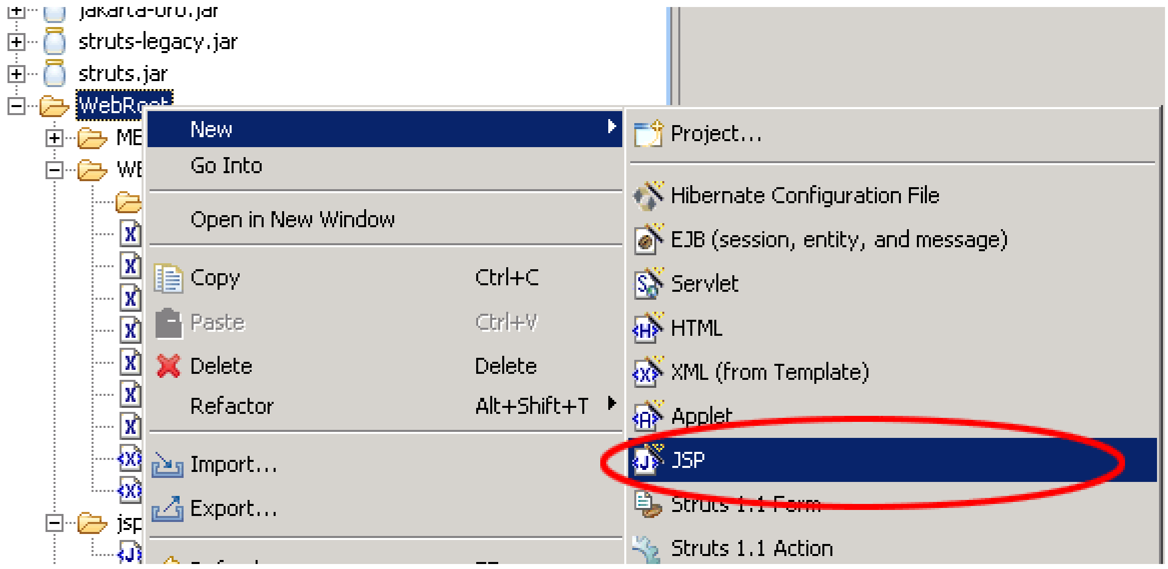
Struts Support for MyEclipse Web Project

Enable project for Struts development

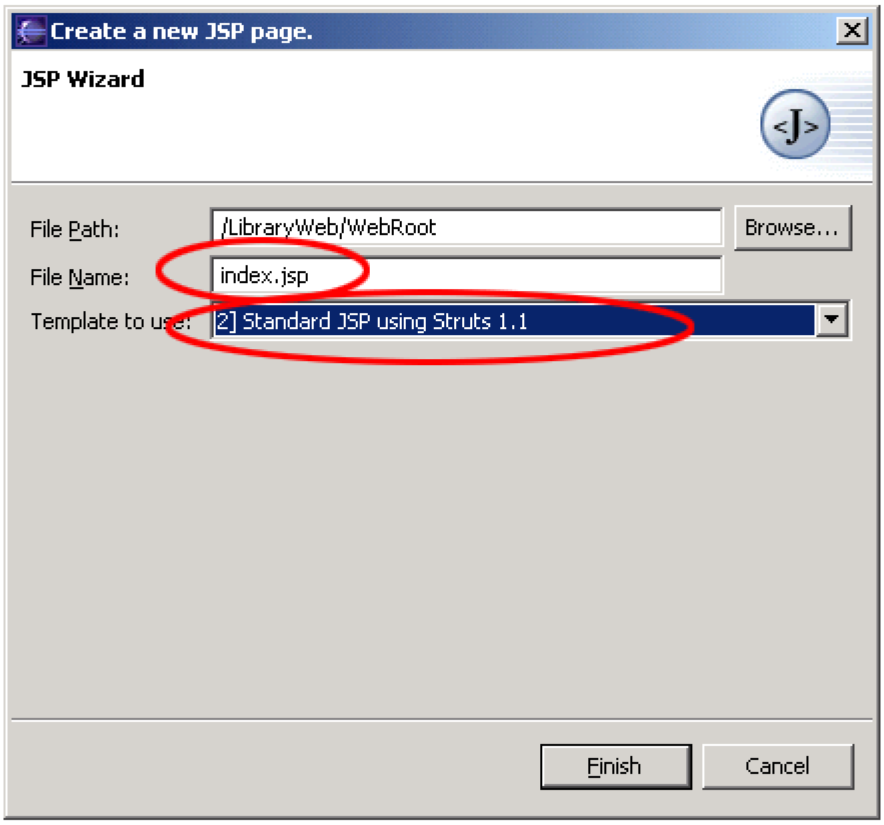
Web project:	LibraryWeb
Web-root folder:	/WebRoot
Servlet specification:	2.4
Struts config path:	/WEB-INF/struts-config.xml
Struts specification:	Struts 1.0 [✓ Struts 1.1]
ActionServlet name:	action
URL pattern	[✓ *.do] /do/*
Base package for new classes:	de.laliluna.tutorial.library.struts
Default application resource:	de.laliluna.tutorial.library.struts.ApplicationResources
	[✓] Install Struts jars [✓] Install Struts TLDs

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`.

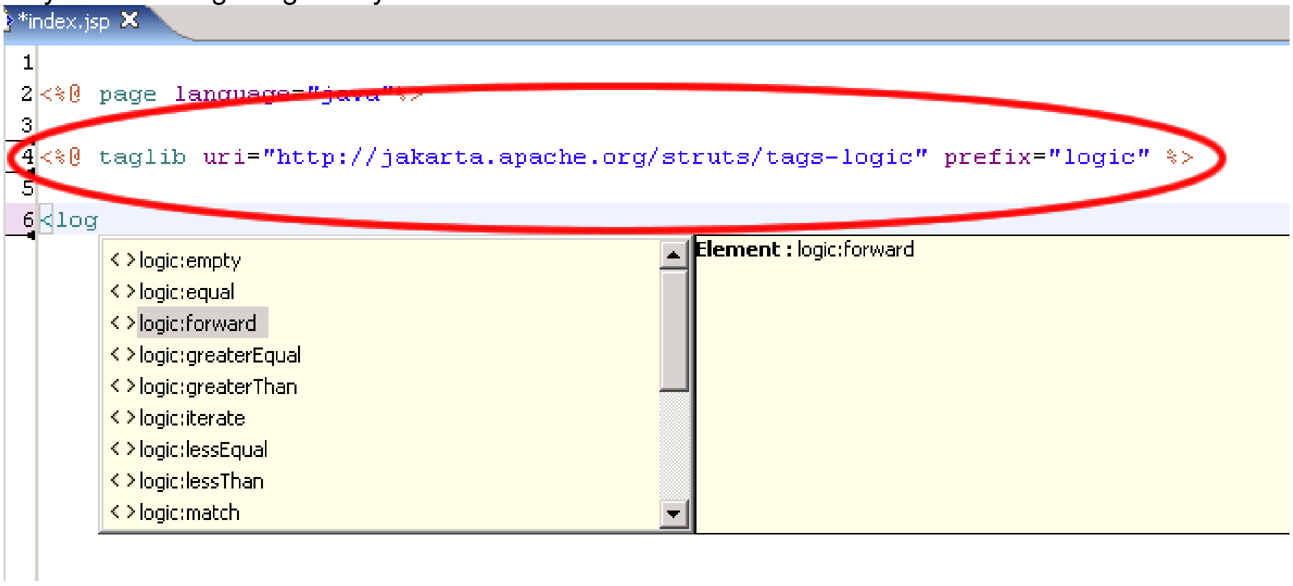


Set the name to `index.jsp` and choose on `template to use` > `Standard JSP using Struts 1.1` MyEcplise will use the template to create the JSP File.



File Path: /LibraryWeb/WebRoot
File Name: index.jsp
Template to use: 2] Standard JSP using Struts 1.1

You will find the file `index.jsp` in the folder `WebRoot` of the project. On the top of the file you will find the struts tag libraries. These includes will be use to access the tags of struts. In your case we only need the logic tag library.



```

<%@ page language="java"%>

<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>

<log
```

Element : logic:forward

Insert the following line below the included logic tag.

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

This line will arranges struts to find a forward with the name `welcome`. If the application don´t find this forward it will leads an error. In the next section i briefly explain the action forward.

Create a second `index.jsp` file in the folder `/WebRoot/jsp`
Change the body of the file to the following

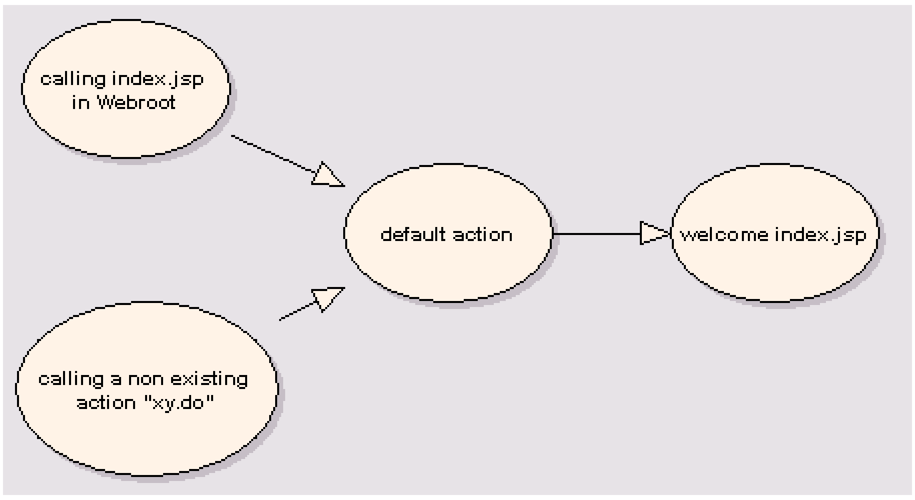
```
<body>
    Welcome!
    <br>
    <html:link action="bookList">Show the booklist</html:link>
</body>
```

Global Action Forwards and Action Mappings

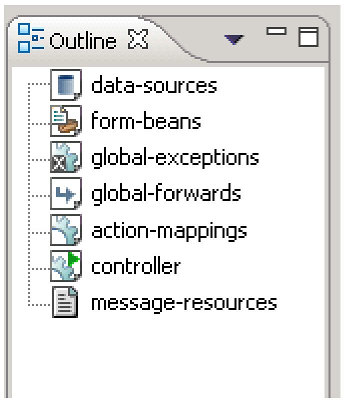
What is an action forward?
A action forward can be used to forward to a jsp or action mapping. There are two different action forwards. The global action forward and the local action forward. You can access a global action forward on each jsp or action class. A local action forward can only be accessed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

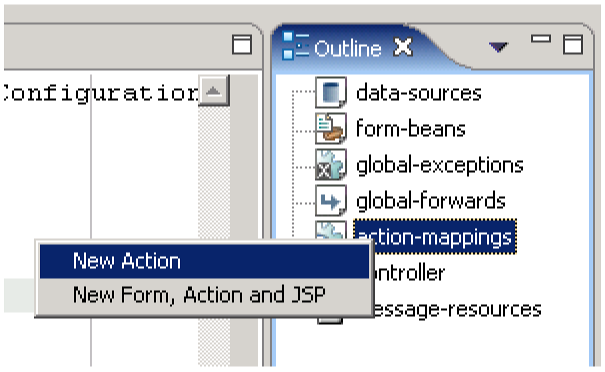
What is a action mapping?
The action mapping is the heart of struts. It managed all actions between the application and the user. You can define which action will be executed by creating a action mapping.

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

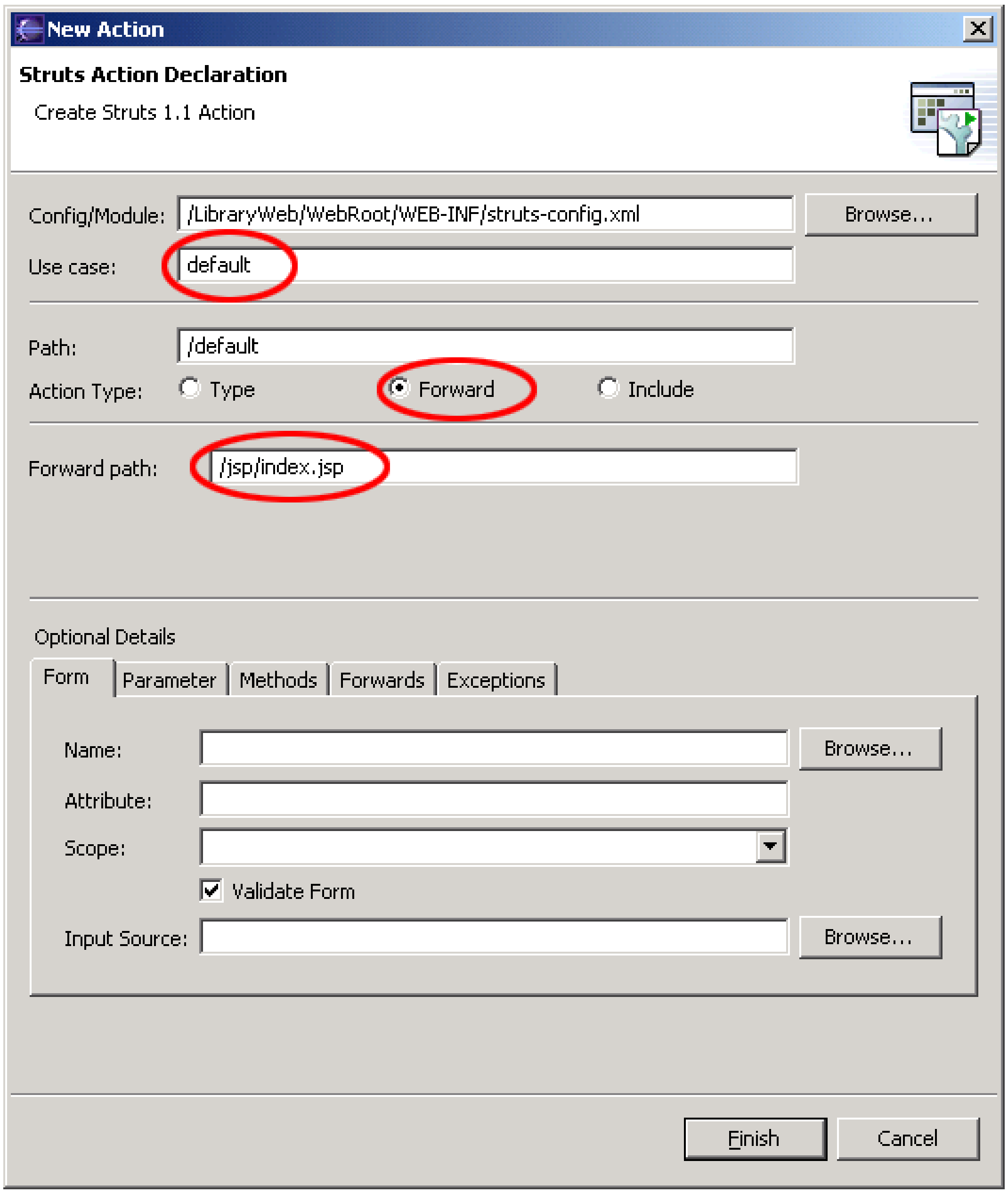

calling index.jsp in Webroot
default action
welcome index.jsp
calling a non existing action "xy.do"

In the first step we create a new action mapping. Open the `struts-config.xml`, you will find it in the folder `WebRoot/WEB-INF`. Right click in the outline window on `action-mapping`.





Choose `Use Case` default and `Action Type` Forward. The `Forward Path` is the welcome page `/jsp/index.jsp`


New Action

Struts Action Declaration

Create Struts 1.1 Action

Config/Module: /LibraryWeb/WebRoot/WEB-INF/struts-config.xml [Browse...]

Use case: default

Path: /default

Action Type: Type [✓ Forward] Include

Forward path: /jsp/index.jsp

Optional Details

Form | Parameter | Methods | Forwards | Exceptions

Name:

Attribute:

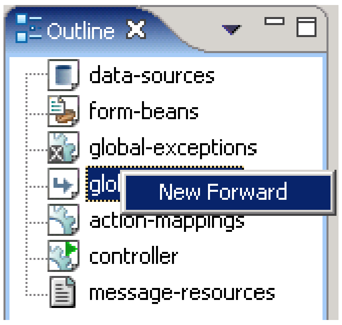
Scope:

[✓ Validate Form]

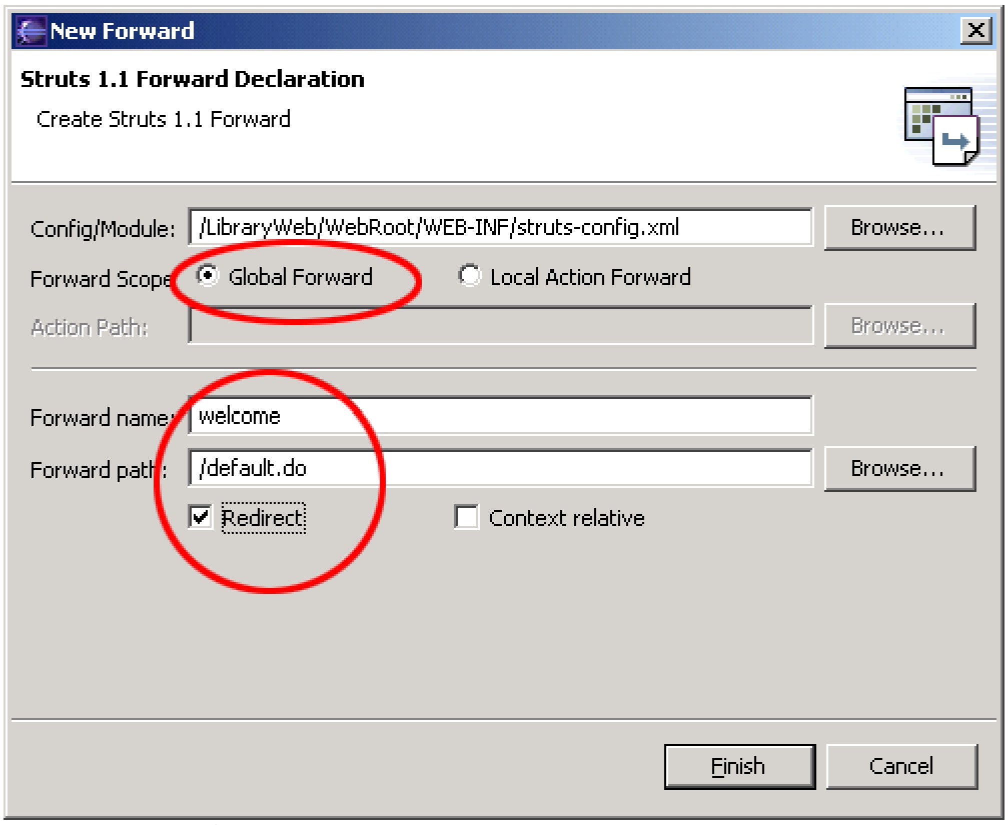
Input Source:

Finish | Cancel

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.


New Forward

Struts 1.1 Forward Declaration

Create Struts 1.1 Forward

Config/Module: /LibraryWeb/WebRoot/WEB-INF/struts-config.xml

Forward Scope: [✓ Global Forward] Local Action Forward

Action Path:

Forward name: welcome

Forward path: /default.do

[✓ Redirect] Context relative

You will see the following in your editor window.

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

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

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

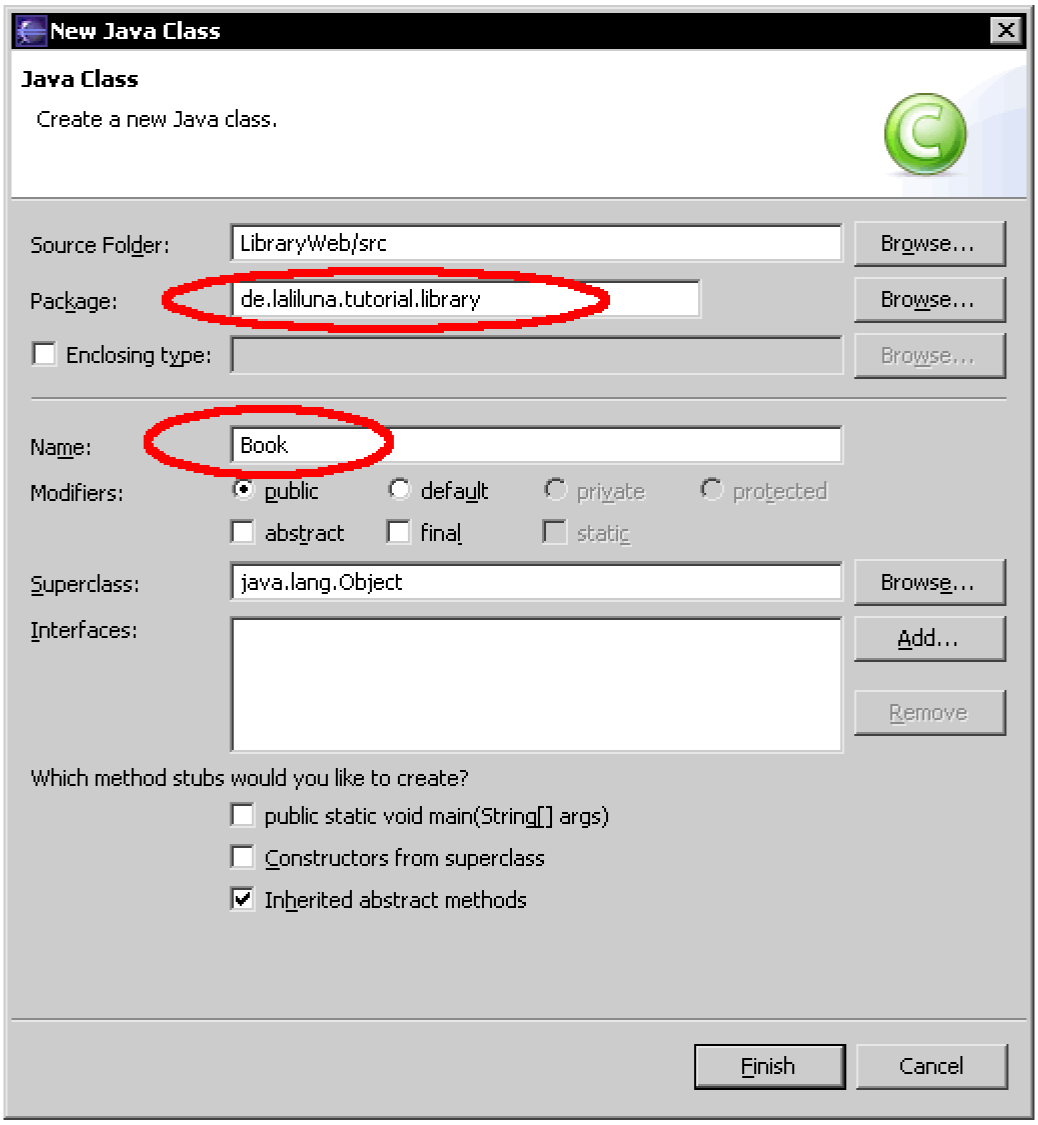
Usecase Book List

A Struts Action does always some kind of business logic and saves the result in a class of type ActionForm. The data from the ActionForm can be displayed in a JSP.

Our Action will read data from the database, save it in the action form. The JSP will display our data.

Create a object class „book“

Create a new class `Book` in the package `de.laliluna.tutorial.library`.


New Java Class

Java Class

Create a new Java class.

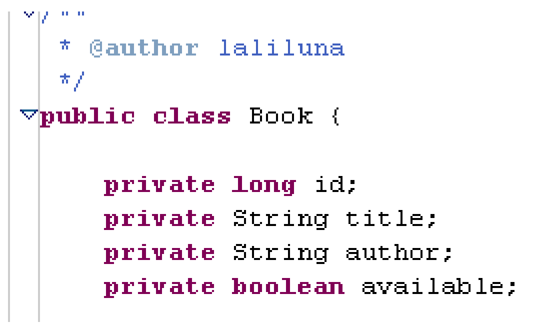
Source Folder:	LibraryWeb/src	Browse...
Package:	de.laliluna.tutorial.library	Browse...
☐ Enclosing type:		Browse...
Name:	Book	
Modifiers:	(●) public ( ) default ( ) private ( ) protected	
	☐ abstract ☐ final ☐ static	
Superclass:	java.lang.Object	Browse...
Interfaces:		Add... Remove

Which method stubs would you like to create?

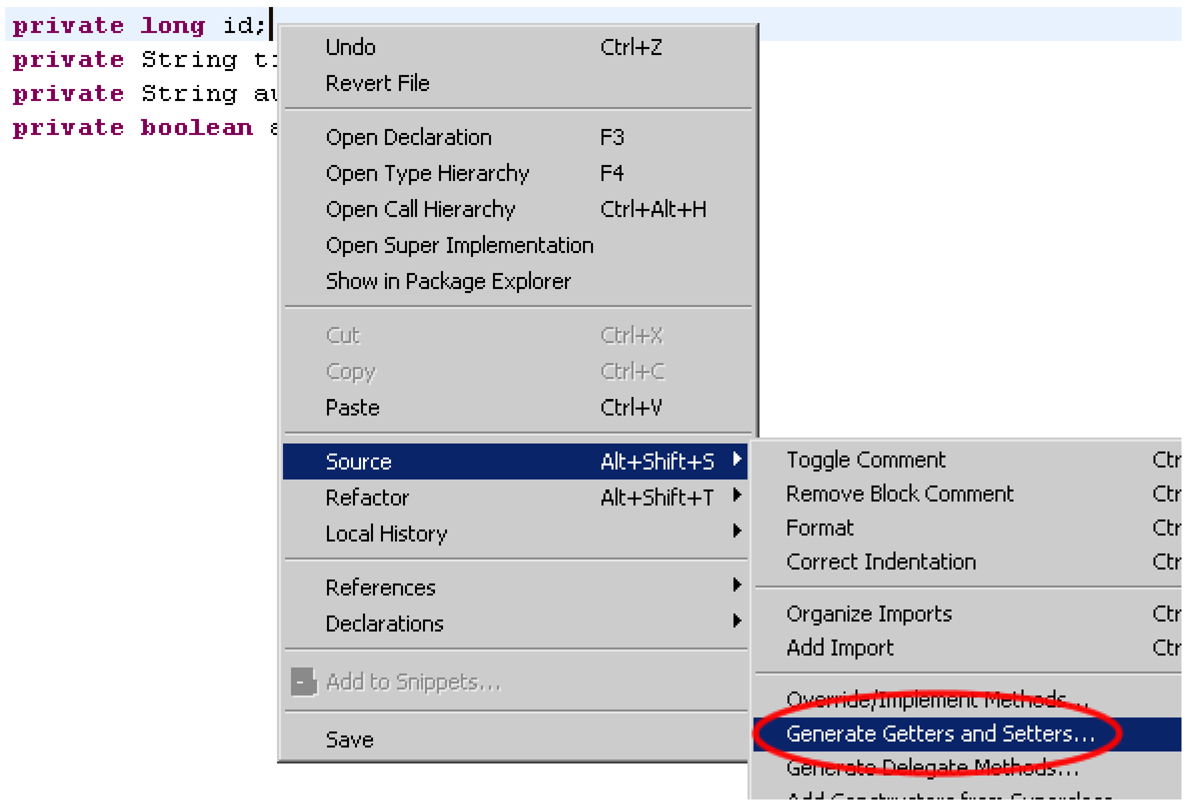
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

Finish Cancel

The Class `Book` represents a book with the properties id, author, title and available. Create four variables.

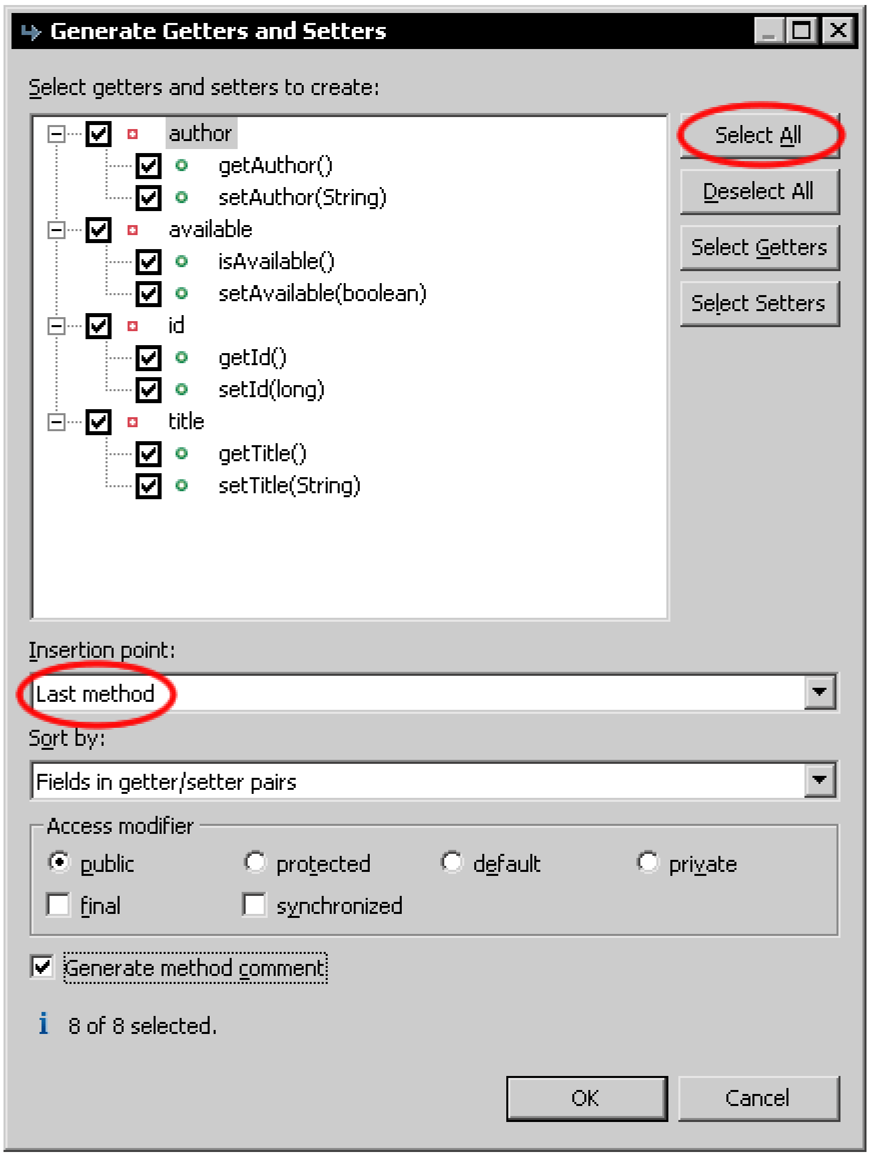

```
/**
 * @author laliluna
 */
public class Book {

    private long id;
    private String title;
    private String author;
    private boolean available;
```

Create a getter and setter for each variable. Right click in your class, `Source > Generate Getters and Setters`



```
private long id;
private String t
private String au
private boolean a
```

Choose `Select All` and insertion point `Last method`.


Generate Getters and Setters

Select getters and setters to create:

- [✓] author
  - [✓] getAuthor()
  - [✓] setAuthor(String)
- [✓] available
  - [✓] isAvailable()
  - [✓] setAvailable(boolean)
- [✓] id
  - [✓] getId()
  - [✓] setId(long)
- [✓] title
  - [✓] getTitle()
  - [✓] setTitle(String)

Insertion point: Last method

Sort by: Fields in getter/setter pairs

Access modifier: [✓ public] protected default private

final: ☐ synchronized: ☐

[✓] Generate method comment

8 of 8 selected.

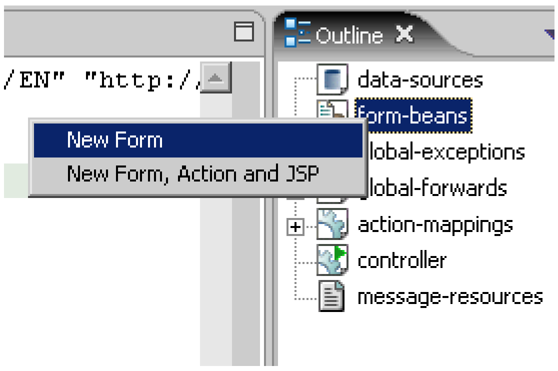
Add two constructors to the class to set the properties on initialisation of the class.

```
// Contructor
public Book(){}

//  Contructor to initial the properties
public Book(long id, String author, String title, boolean available) {
      this.id = id;
      this.author = author;
      this.title = title;
      this.available = available;
}
```

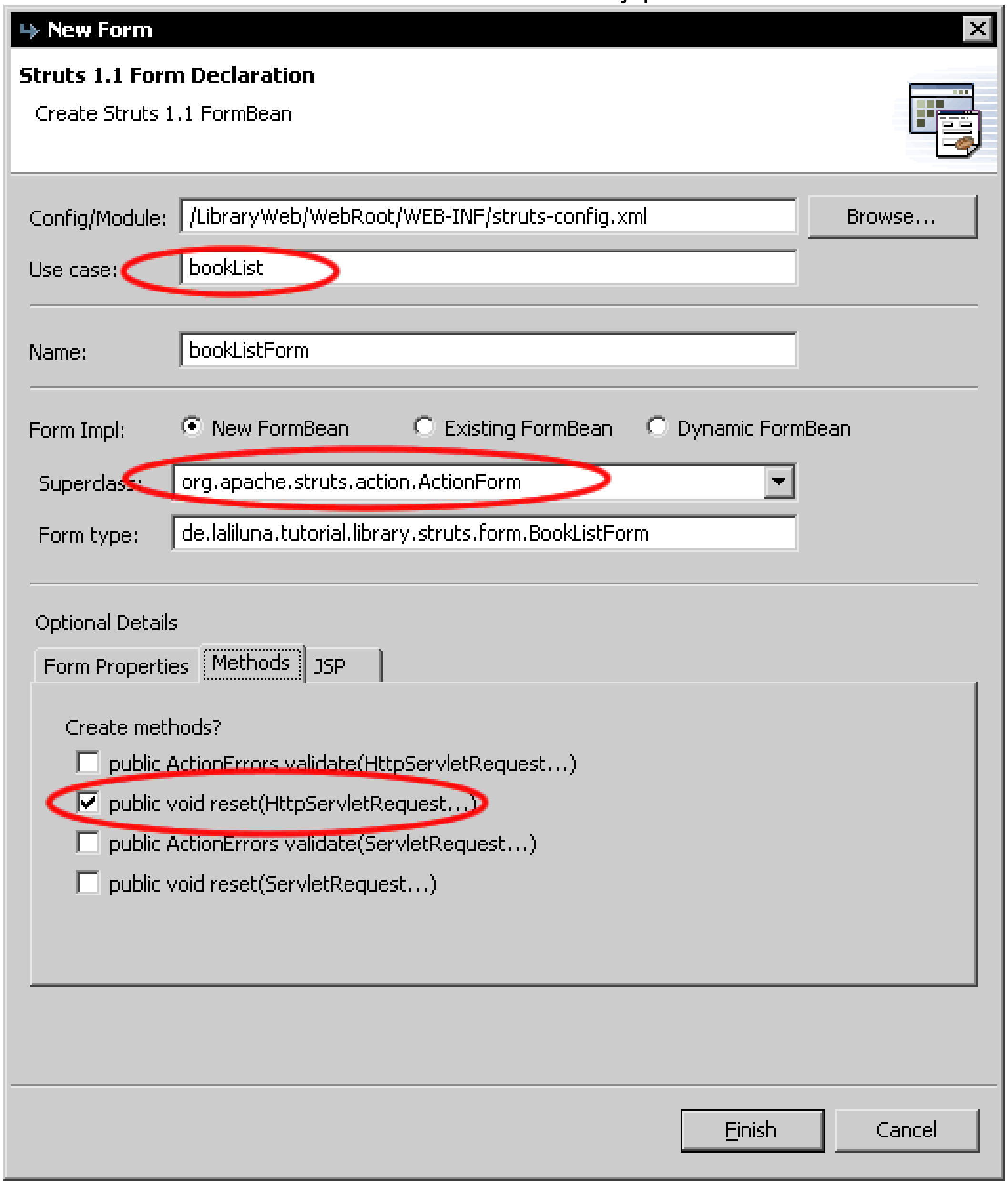
Thats all for our book class!

Create a form bean, action form and jsp

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



Use Case is `bookList`, Superclass `org.apache.struts.ActionForm`. Select only `public void reset..` on methods. Set the name of the jsp file on JSP.



New Form

Struts 1.1 Form Declaration

Create Struts 1.1 FormBean

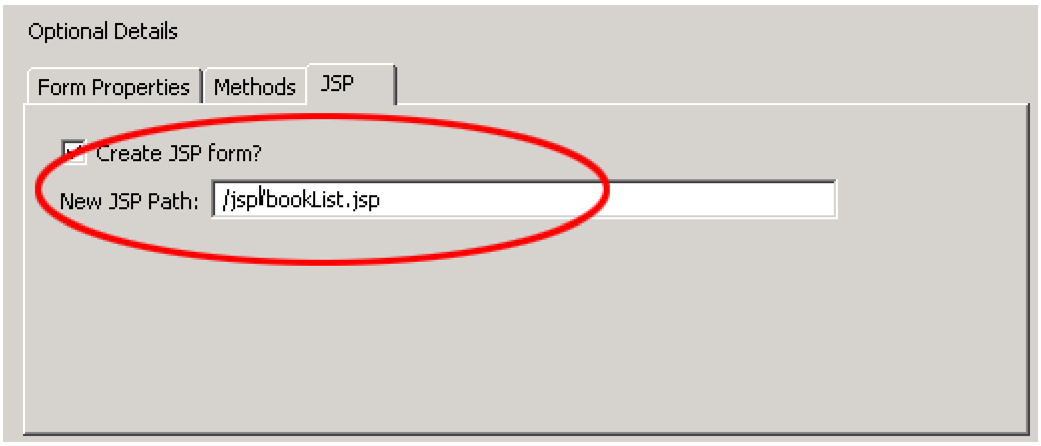
Config/Module:	/LibraryWeb/WebRoot/WEB-INF/struts-config.xml
Use case:	bookList
Name:	bookListForm
Form Impl:	[✓ New FormBean] Existing FormBean Dynamic FormBean
Superclass:	org.apache.struts.action.ActionForm
Form type:	de.laliluna.tutorial.library.struts.form.BookListForm

Optional Details

Form Properties | Methods | JSP

Create methods?

- [ ] public ActionErrors validate(HttpServletRequest...)
- [✓] public void reset(HttpServletRequest...)
- [ ] public ActionErrors validate(ServletRequest...)
- [ ] public void reset(ServletRequest...)

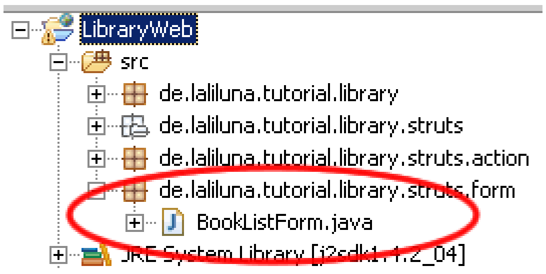

Optional Details

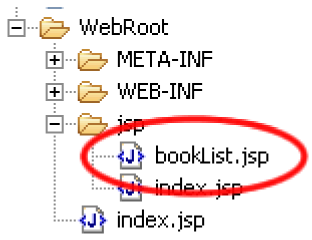
Form Properties | Methods | JSP

Create JSP form? ✓

New JSP Path: /jsp/bookList.jsp

The package explorer looks like the pictures below.


```
LibraryWeb
  src
    de.laliluna.tutorial.library
    de.laliluna.tutorial.library.struts
    de.laliluna.tutorial.library.struts.action
    de.laliluna.tutorial.library.struts.form
      BookListForm.java
  JRE System Library [j2sdk1.4.2_04]
```


```
WebRoot
  META-INF
  WEB-INF
  jsp
    bookList.jsp
    index.jsp
  index.jsp
```

Edit the source code of the action form class

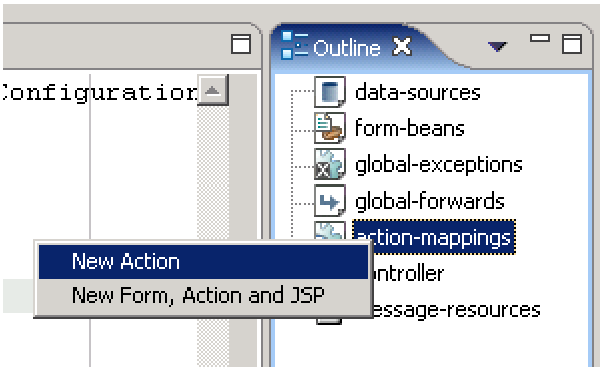
Open the file `BookListForm.java` and add the following soure code.

```
public class BookListForm extends ActionForm {

	private Collection books;

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

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

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

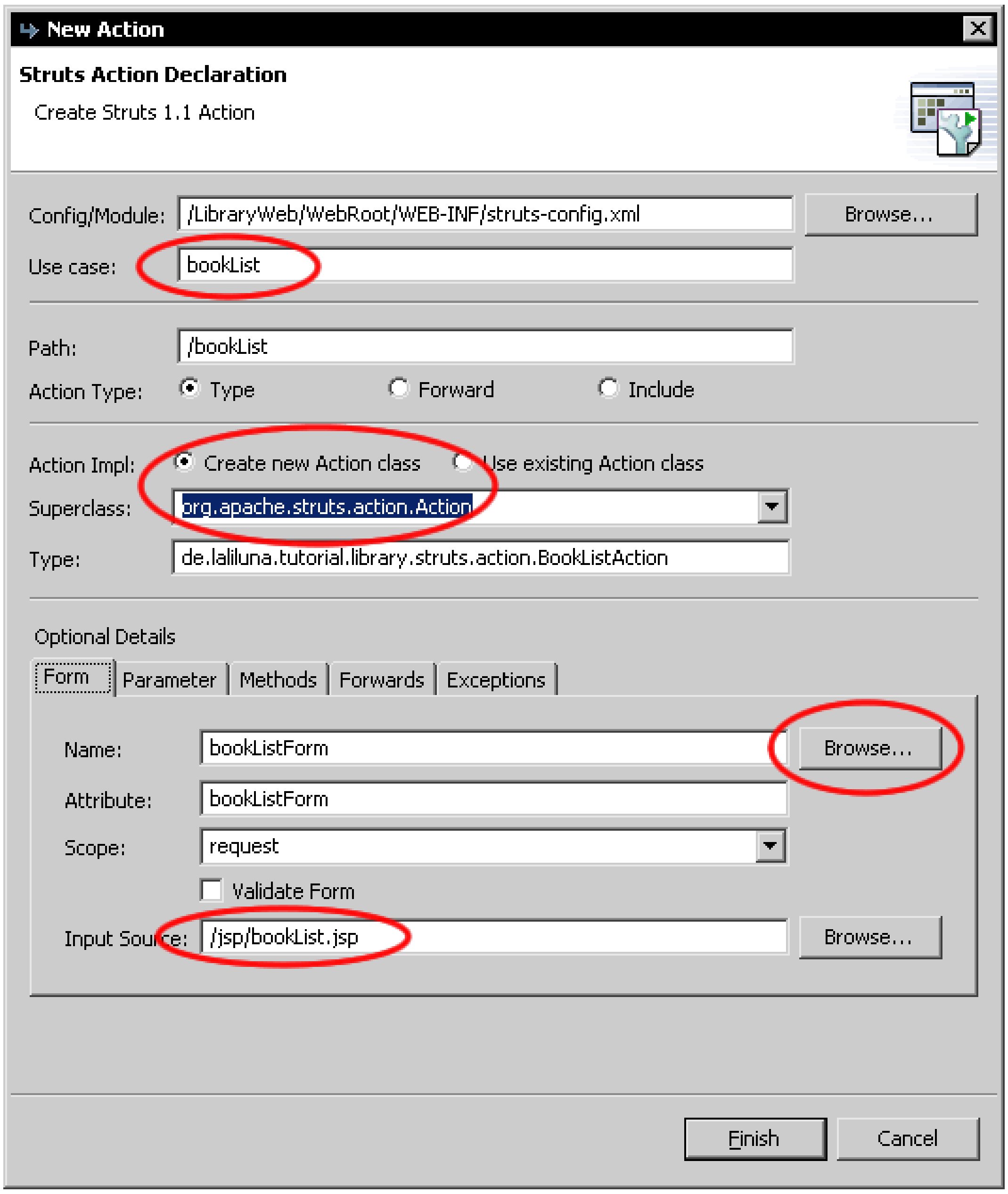
Define a collection `books` and generate a getter and setter. In your `reset` method initial the collection with an array list.

Create an action mapping and action class

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`



New Action

Struts Action Declaration

Create Struts 1.1 Action

Config/Module: /LibraryWeb/WebRoot/WEB-INF/struts-config.xml

Use case: bookList

Path: /bookList

Action Type: [✓ Type] Forward Include

Action Impl: [✓ Create new Action class] Use existing Action class

Superclass: org.apache.struts.action.Action

Type: de.laliluna.tutorial.library.struts.action.BookListAction

Optional Details

Form | Parameter | Methods | Forwards | Exceptions

Name: bookListForm

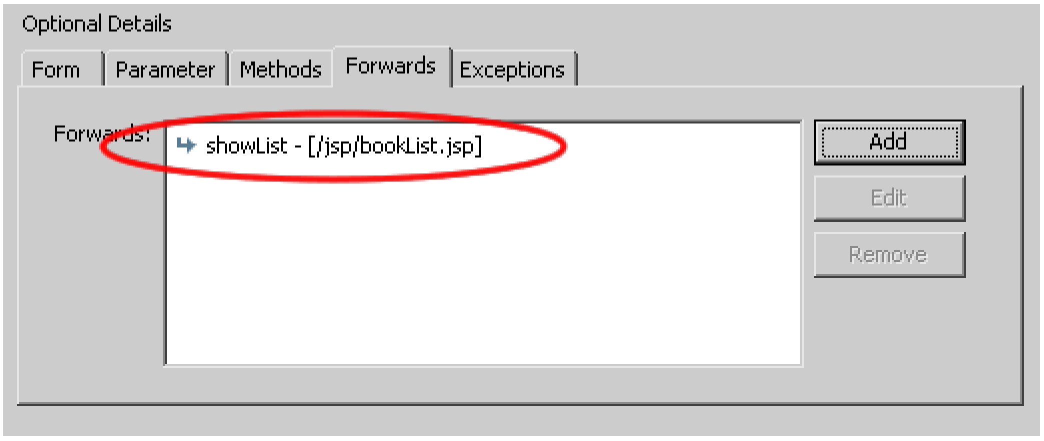
Attribute: bookListForm

Scope: request

Validate Form: ☐

Input Source: /jsp/bookList.jsp

Now add a forward `showList` to the action mapping.



Optional Details

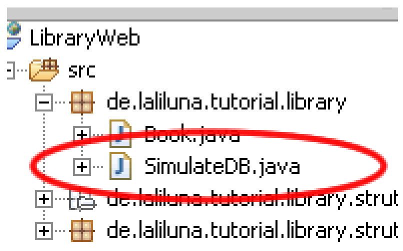
Form | Parameter | Methods | Forwards | Exceptions

Forwards: showList - [/jsp/bookList.jsp]

You will find the action class `bookListAction` in your package `de.laliluna.tutorial.library.action.`

Class to provide test data

We do not use a database in this tutorial and want some test data. Open the source code we provided as download and copy the class `simulateDB.java` in your package `de.laliluna.tutorial.library.`


LibraryWeb
src
de.laliluna.tutorial.library
Book.java
SimulateDB.java
de.laliluna.tutorial.library.strut
de.laliluna.tutorial.library.strut

Edit the source code of the action class

Open the class bookListAction and edit the method execute. The command `mapping.findForward(„showList“)` will search for a local forward with the name `showList`

```
/**
 * 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;


   /* lalinuna.de 03.11.2004
    * init SimulateDB class and set some dummy data
    */
    SimulateDB simulateDB = new SimulateDB();
    bookListForm.setBooks(simulateDB.getAllBooks(request.getSession()));


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

Yeah thats 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.

Output the test data on the jsp file

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

```
<%@ 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>
<table border="1">
      <tbody>
            <%-- set the header --%>
            <tr>
                  <td>Author</td>
                  <td>Book name</td>
                  <td>Available</td>
                  <td>&nbsp;</td>
                  <td>&nbsp;</td>
            </tr>
            <%-- check if book exists and display message or iterate over books
--%>
            <logic:empty name="bookListForm" property="books">
                  <tr>
                        <td colspan="5">No books available</td>
                  </tr>
            </logic:empty>
            <logic:notEmpty name="bookListForm" property="books">
                  <logic:iterate name="bookListForm" property="books" id="book">
                        <tr>
                              <%-- print out the 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>

                              <%-- print out the edit and delete link for each
book --%>
                              <td><html:link action="bookEdit.do?do=editBook"
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>
                        </tr>
                  </logic:iterate>
            </logic:notEmpty>

            <%-- end interate --%>
      </tbody>
</table>
</body>
</html>
```

The tag <logic:iterate> loops over the collection `books` of the form bean `bookListForm`
Within the tag `<logic:iterate>` you have access to the properties of the book. The tag `<bean:write>` prints out a property (author, title) on the current position.
`<html:checkbox>` creates a checkbox.

Very good. If you like you do a first test of your application right here. Have a look at the end of the tutorial, to see how to test.

Usecase Add, edit and remove the data

In the next section we add the functionality to add, edit and remove the data.

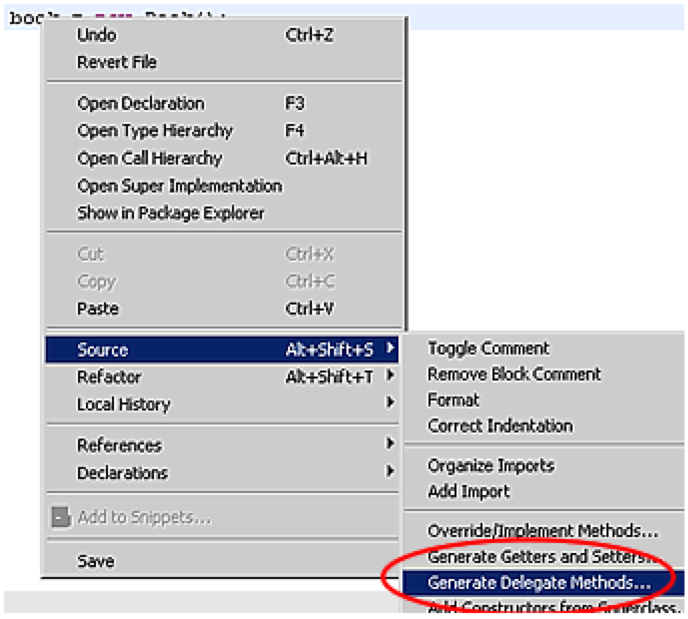
New form bean

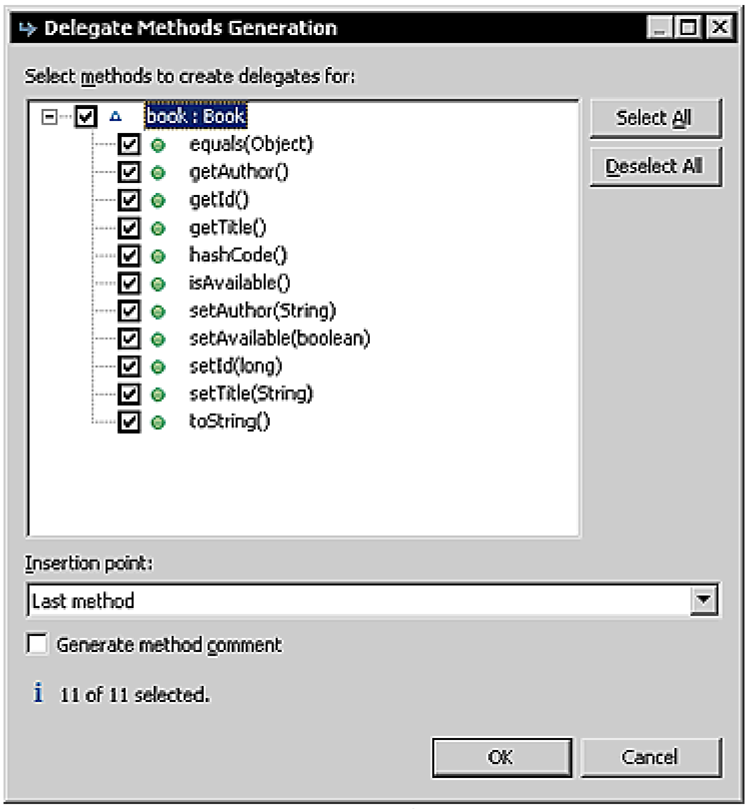
Create a new form bean and action form class. Set Use case to bookEdit and remove all methods on `Optional details – Methods`. Let MyEcplise create the jsp file for us.

Open the class `BookEditForm.java` in `de.laliluna.tutorial.library.form`.
Create a new instance `book` of the class `Book`

```
Book book = new Book();
```

Generate a getter and setter and delegate all methods of the class Book.





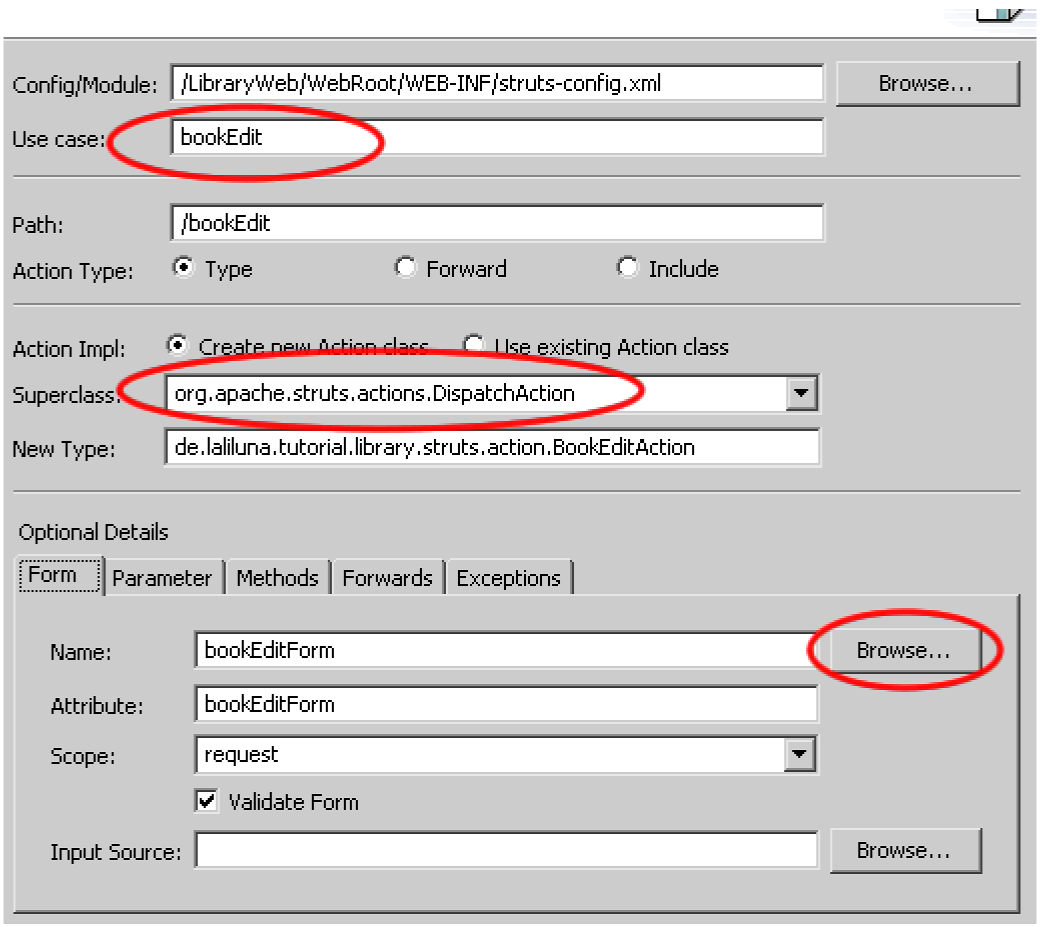
The source code looks like the following.

```
public class BookEditForm extends ActionForm {
	Book book = new Book();

	public Book getBook() {
		return book;
	}
	public void setBook(Book book) {
		this.book = book;
	}
	public boolean equals(Object arg0) {
		return book.equals(arg0);
	}
	public String getAuthor() {
		return book.getAuthor();
	}
	public long getId() {
		return book.getId();
	}
	public String getTitle() {
		return book.getTitle();
	}
	public int hashCode() {
		return book.hashCode();
	}
	public boolean isAvailable() {
		return book.isAvailable();
	}
	public void setAuthor(String author) {
		book.setAuthor(author);
	}
	public void setAvailable(boolean available) {
		book.setAvailable(available);
	}
	public void setId(long id) {
		book.setId(id);
	}
	public void setTitle(String title) {
		book.setTitle(title);
	}
	public String toString() {
		return book.toString();
	}
}
```

The class Book is set in the action form class and we have access to the properties.

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`.



Config/Module: /LibraryWeb/WebRoot/WEB-INF/struts-config.xml

Use case: bookEdit

Path: /bookEdit

Action Type: [✓ Type] Forward Include

Action Impl: [✓ Create new Action class] Use existing Action class

Superclass: org.apache.struts.actions.DispatchAction

New Type: de.laliluna.tutorial.library.struts.action.BookEditAction

Optional Details

Form: Name: bookEditForm

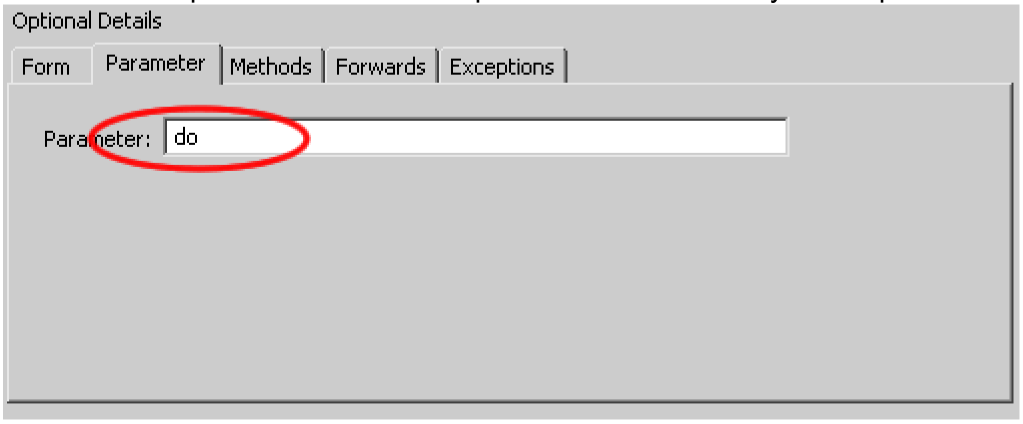
Attribute: bookEditForm

Scope: request

[✓] Validate Form

Input Source:

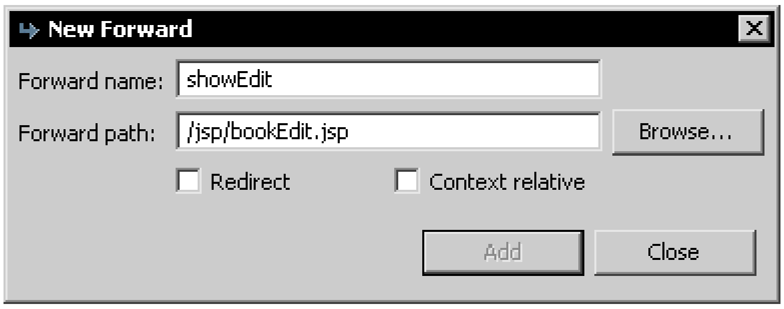
On Parameter we add a parameter `do`. These parameter is needed by the dispatch action class.



Optional Details

Parameter: do

Add three new forwards. One is for the edit page, the second for the add page, where you can add the books and the last forward redirect the user to the book listing.

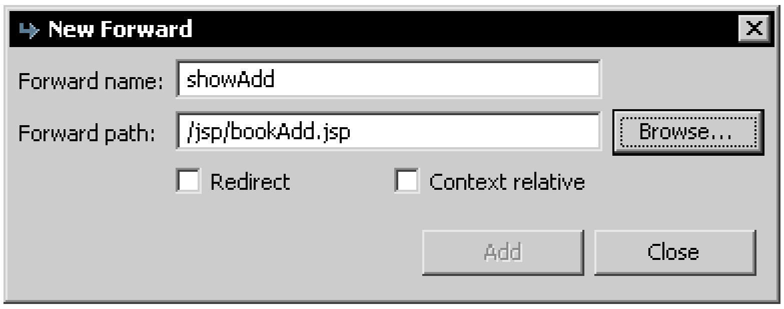


New Forward

Forward name: showEdit

Forward path: /jsp/bookEdit.jsp

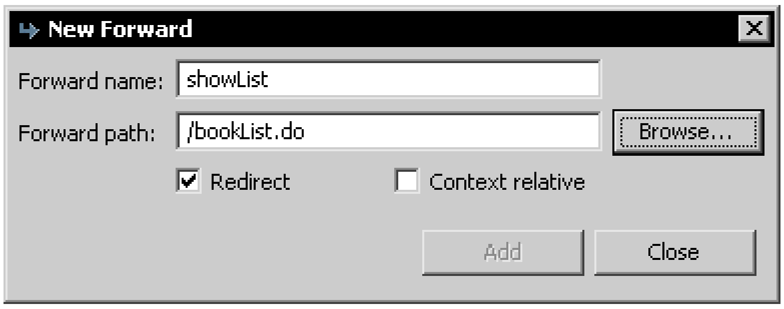
Redirect: ☐ Context relative: ☐


New Forward

Forward name: showAdd

Forward path: /jsp/bookAdd.jsp

☐ Redirect ☐ Context relative


New Forward

Forward name: showList

Forward path: /bookList.do

☑ Redirect ☐ Context relative

The last forward is different to the others. It refers to an existing action mapping and redirect the user.

Now create a new jsp file `bookAdd.jsp` in the folder `/WebRoot/jsp`. The forward `showAdd` forwards to this page.

Add the source code to the jsp files

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

```
<%@ 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>
				<tr>
					<td colspan="2">
						<html:submit>Save</html:submit>
					</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.

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

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

```
<%-- hidden field that contains the id of the book --%>
<html:hidden property="id" />
```

Source code of the dispatch action class

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

```
/**
 * Method editBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
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
 */
 long id = Long.parseLong(request.getParameter("id"));

 /* lalinuna.de 04.11.2004
 * init SimulateDB class and get book by id
 */
 SimulateDB simulateDB = new SimulateDB();
 bookEditForm.setBook(simulateDB.loadBookById(id, request.getSession()));

 return mapping.findForward("showEdit");

}
```

The method `editBook` get the parameter `id` of the `request` and reads the book by id from the simulated database. The forward `showEdit` refres to the edit page `bookEdit.jsp`

```
/**
 * Method deleteBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward deleteBook(
      ActionMapping mapping,
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response) {
      BookEditForm bookEditForm = (BookEditForm) form;

      /* lalinuna.de 04.11.2004
      * get id of the book from request
      */
      long id = Long.parseLong(request.getParameter("id"));

      /* lalinuna.de 04.11.2004
      * init SimulateDB class and delete book by id
      */
      SimulateDB simulateDB = new SimulateDB();
      simulateDB.deleteBookById(id, request.getSession());

      return mapping.findForward("showList");

}
```

The method `deleteBook` get the parameter `id` of the `request` and remove the book by id from the simulated database. The forward `showList` refres to the book listing page `bookList.jsp`

```
/**
 * Method addBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward addBook(
      ActionMapping mapping,
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response) {
      BookEditForm bookEditForm = (BookEditForm) form;

      return mapping.findForward("showAdd");

}
```

The method `addBook` forwards on the add page `bookAdd.jsp`

```
/**
 * Method saveBook
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return ActionForward
 */
public ActionForward saveBook(
      ActionMapping mapping,
      ActionForm form,
      HttpServletRequest request,
      HttpServletResponse response) {
      BookEditForm bookEditForm = (BookEditForm) form;

      /* lalinuna.de 04.11.2004
      * init SimulateDB class and get data by id
      */
      SimulateDB simulateDB = new SimulateDB();
      simulateDB.saveToDB(bookEditForm.getBook(), request.getSession());

      return mapping.findForward("showList");

}
```

The last method get the book of the form bean `bookEditForm` and save it in the simulated

Database.

Edit the book listing page

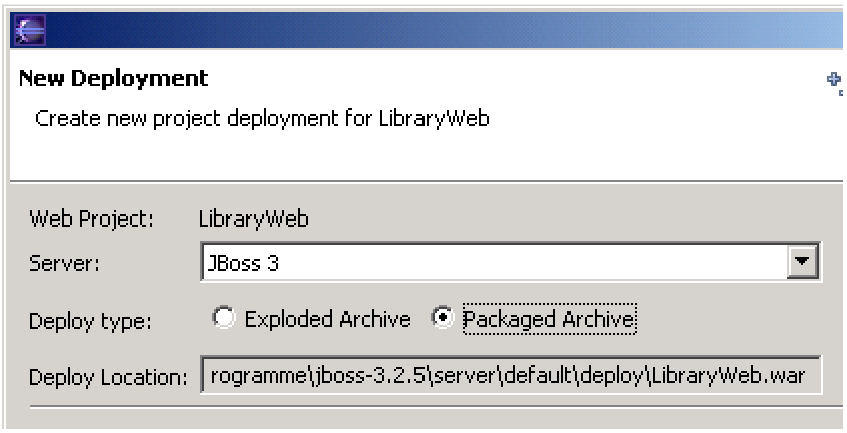
Open the file `bookList.jsp` and change the source code.

```
<%@ 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>
<table border="1">
    <tbody>
    <%--  set the header --%>
    <tr>
       <td>Author</td>
       <td>Book  name</td>
       <td>Available</td>
       <td>&nbsp;</td>
       <td>&nbsp;</td>
    </tr>
    <%-- check if book exists and display message or iterate over books  --%>
  <logic:empty name="bookListForm" property="books">
    <tr>
       <td colspan="5">No books available</td>
    </tr>
  </logic:empty>
  <logic:notEmpty name="bookListForm" property="books">
    <logic:iterate name="bookListForm" property="books" id="book">
       <tr>
       <%-- print out the 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>
       <%-- print out the edit and delete link for each book --%>
       <td><html:link action="bookEdit.do?do=editBook" 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>
       </tr>
       </logic:iterate>
    </logic:notEmpty>

    <%-- print out the add link --%>
    <tr>
       <td colspan="5"><html:link action="bookEdit.do?do=addBook">Add a new
book</html:link>
       </td>
    </tr>
    <%-- end interate --%>
  </tbody>
</table>
</body>
</html>
```

Congratulation, you have finished a simple library application. Now you can test the library.

Test the application

Start the jboss and deploy the project as package archiv.



New Deployment

Create new project deployment for LibraryWeb

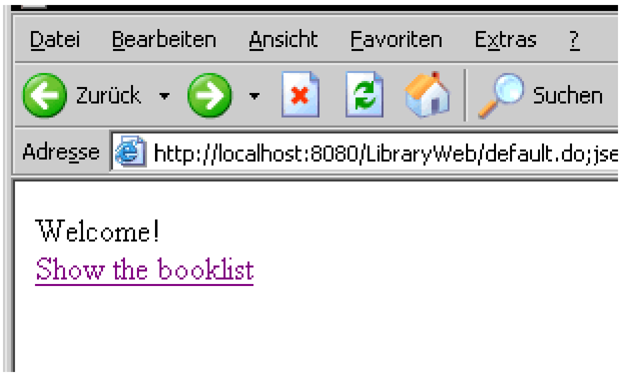
Web Project: LibraryWeb

Server: JBoss 3

Deploy type: Exploded Archive [✓ Packaged Archive]

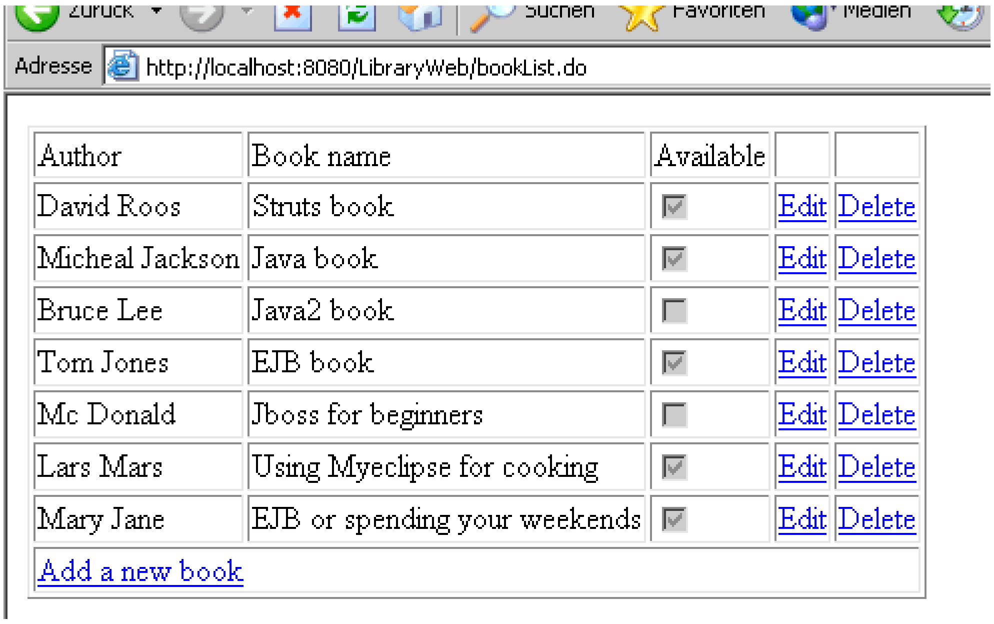
Deploy Location: rogramme\jboss-3.2.5\server\default\deploy\LibraryWeb.war

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



http://localhost:8080/LibraryWeb/default.do;jse

Welcome!
Show the booklist



http://localhost:8080/LibraryWeb/bookList.do

Author	Book name	Available		
David Roos	Struts book	✓	Edit	Delete
Micheal Jackson	Java book	✓	Edit	Delete
Bruce Lee	Java2 book	☐	Edit	Delete
Tom Jones	EJB book	✓	Edit	Delete
Mc Donald	Jboss for beginners	☐	Edit	Delete
Lars Mars	Using Myeclipse for cooking	✓	Edit	Delete
Mary Jane	EJB or spending your weekends	✓	Edit	Delete
Add a new book				

Thats all !!