Java Server Pages combined with servlets in action

We want to create a small web application (library), that illustrates the usage of JavaServer Pages combined with Java Servlets. We use the JavaServer Pages to display data (presentation) and the servlets to control and call the business logic of the web application.

First we shortly explain the characteristics of these two technologies.

Generals

Author:
Sascha Wolski
http://www.laliluna.de/tutorials.html
Tutorials for Struts, EJB, xdoclet and eclipse.
Date:
September, 23th 2004

Source code:
http://www.laliluna.de/assets/tutorials/java-servlets-jsp-tutorial.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/java-servlets-jsp-tutorial-en.pdf

Development Tools
Eclipse 3.x

Optional:
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.)

Application Server
Jboss 3.2.5
You may use Tomcat here if you like.

Java Servlets

Servlets represent java programs that run on a web server. They allow the developer to produce dynamic web sites with java.

A Servlet has the following tasks

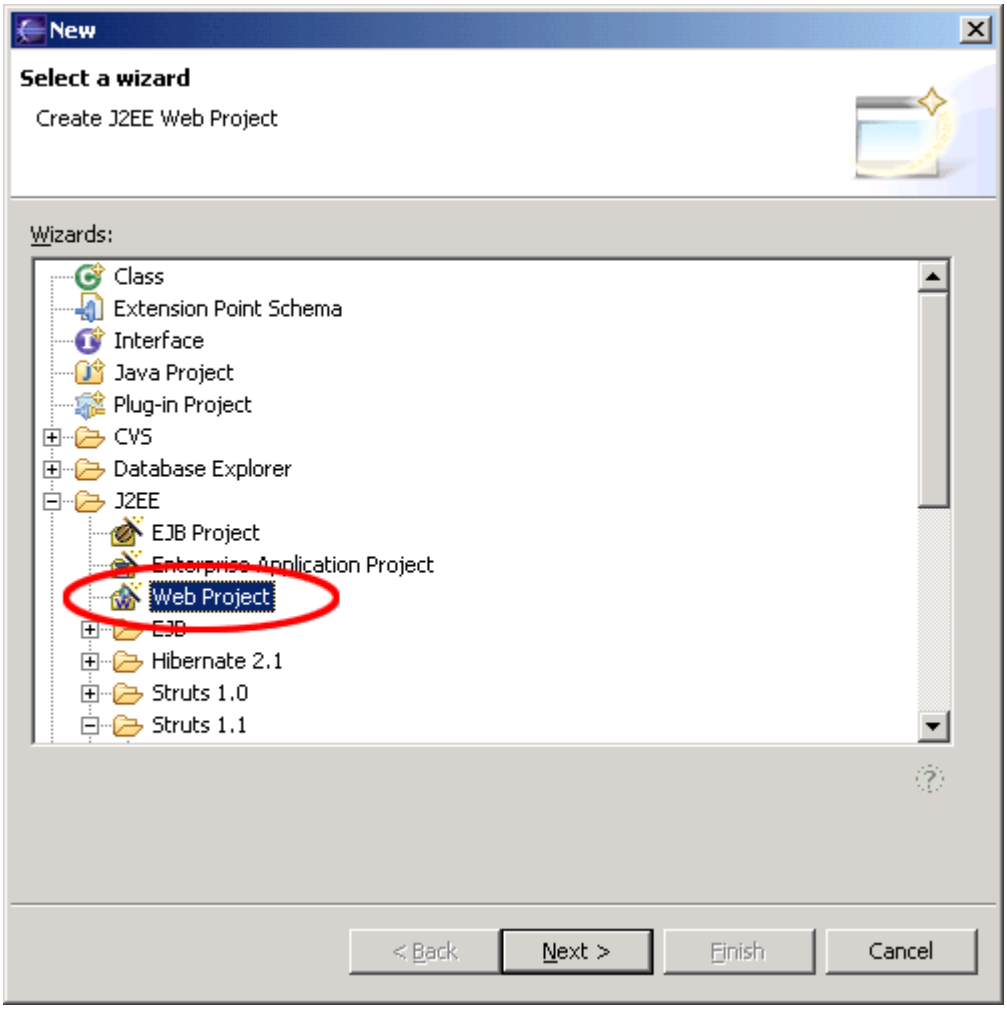
- It reads and processes data, which a user inputs in a HTML form on a web page.
- Other informations can be processed, e.g. what browser or system will be used.
- It generate results with the existing data, calls the business logic directly in the servlet or another class, which contains the logic or executes a database query.
- The results are formated. If the Browser expects an answer in the HTML format, then the results must be formatted in accordance with the standard. It is possible to return different formats of data with a servlet. (gif, jpeg, doc, etc.).
- Suitable answer parameters are set. Befor the servlet return the data to the browser, it sends some parameter. The parameter contains the format, that will returned by the servlet, what time the browser use to cache the site and some more.

Java Server Pages (JSP)

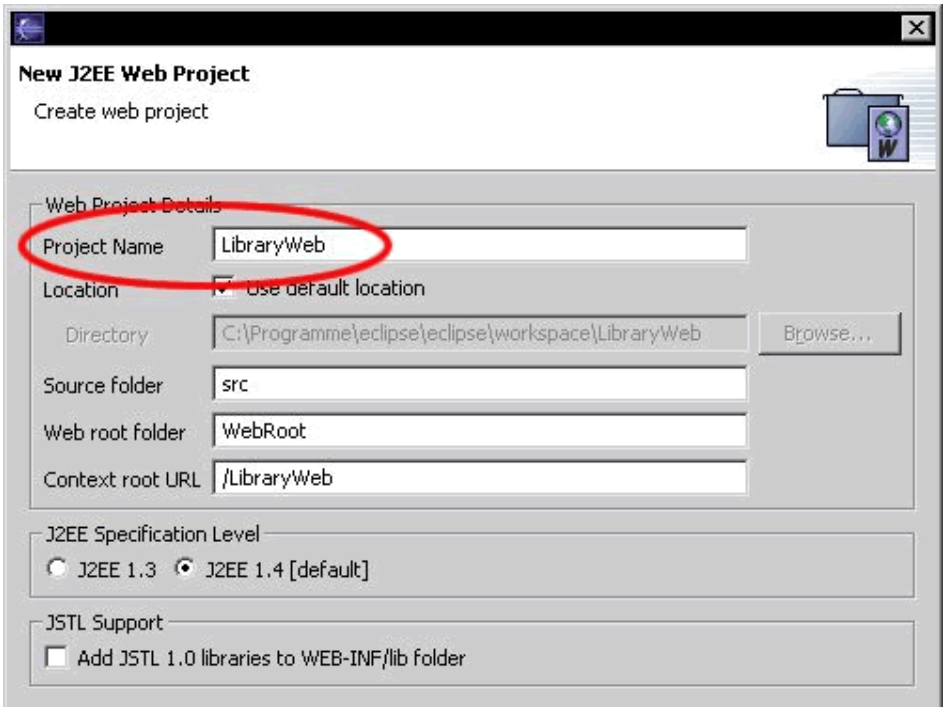
JavaServer Pages (JSP) are text documents, which are similar to HTML Files. But you find also java code in the JSP File. JavaServer Pages allow you to mix regular static HTML with dynamic generated contents of Java code. The java code is inserted in the HTML document on a JSP File, differently to a servlet, where the HTML code is embedded in the java code.

Create a new Web Projekt

Right click on the package explorer and choose `New > Project`.
Choose the wizard Web Project in J2EE.



Set a nice name for your project.



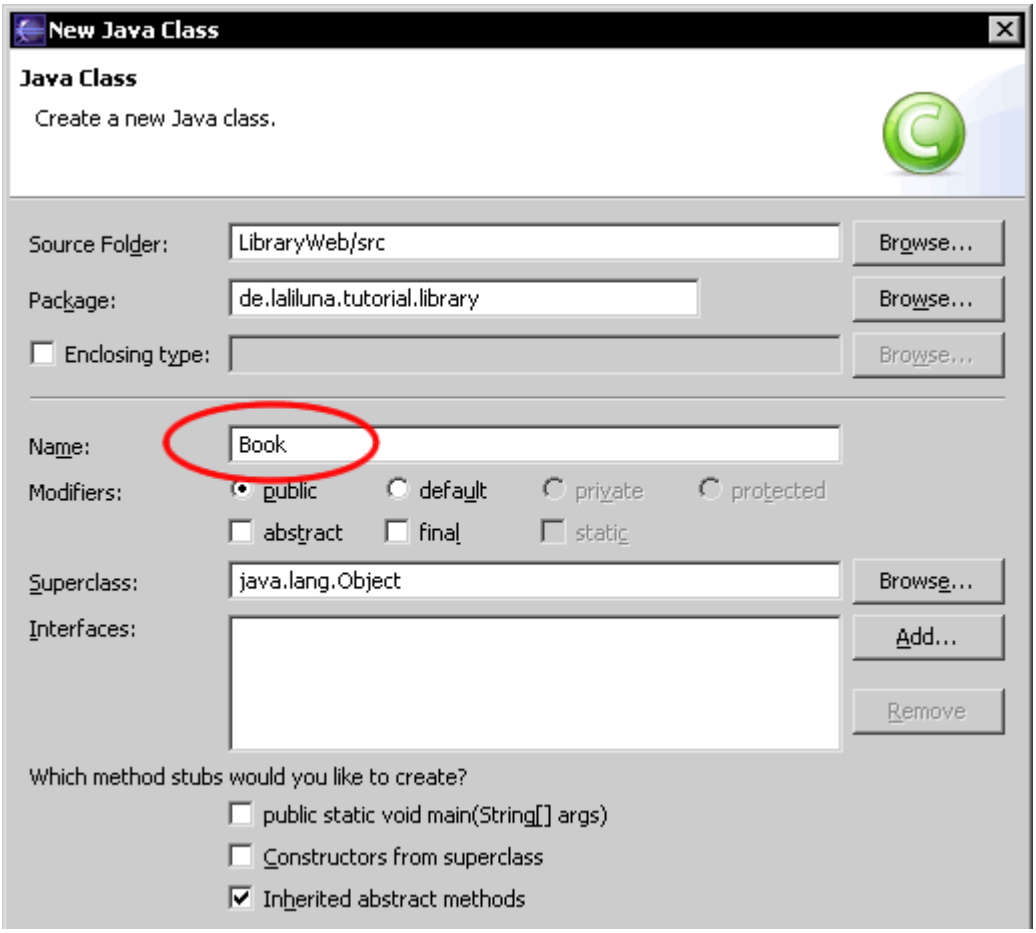
Class with dummy data

In this example we do not use a database binding. We use a class which provide the dummy data and some methods for add, update, delete. Create a new Package `de.laliluna.tutorial.library` in `src` of the project. Download the complete project an copy the file `SimulatedDB.java` from the folder `/de/laliluna/tutorial/library` in the package.

Create the class book

You can borrow a book in a library, so we need a class that represent the book.

Create a package `de.laliluna.tutorial.library` and a new class `New > Class`.



The object class book contains the following private properties as variables.

- id
- title
- author
- available

We create for the properties getter- and setter-methods, to access the private properties.

Open the class Book and add the properties and getter- and setter-methods. Add constructors to this class.

```
public class Book {

    private long id = 0;
    private String title = "";
    private String author = "";
    private boolean available = false;

    // 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;
    }

    public boolean isAvailable() {
        return available;
```

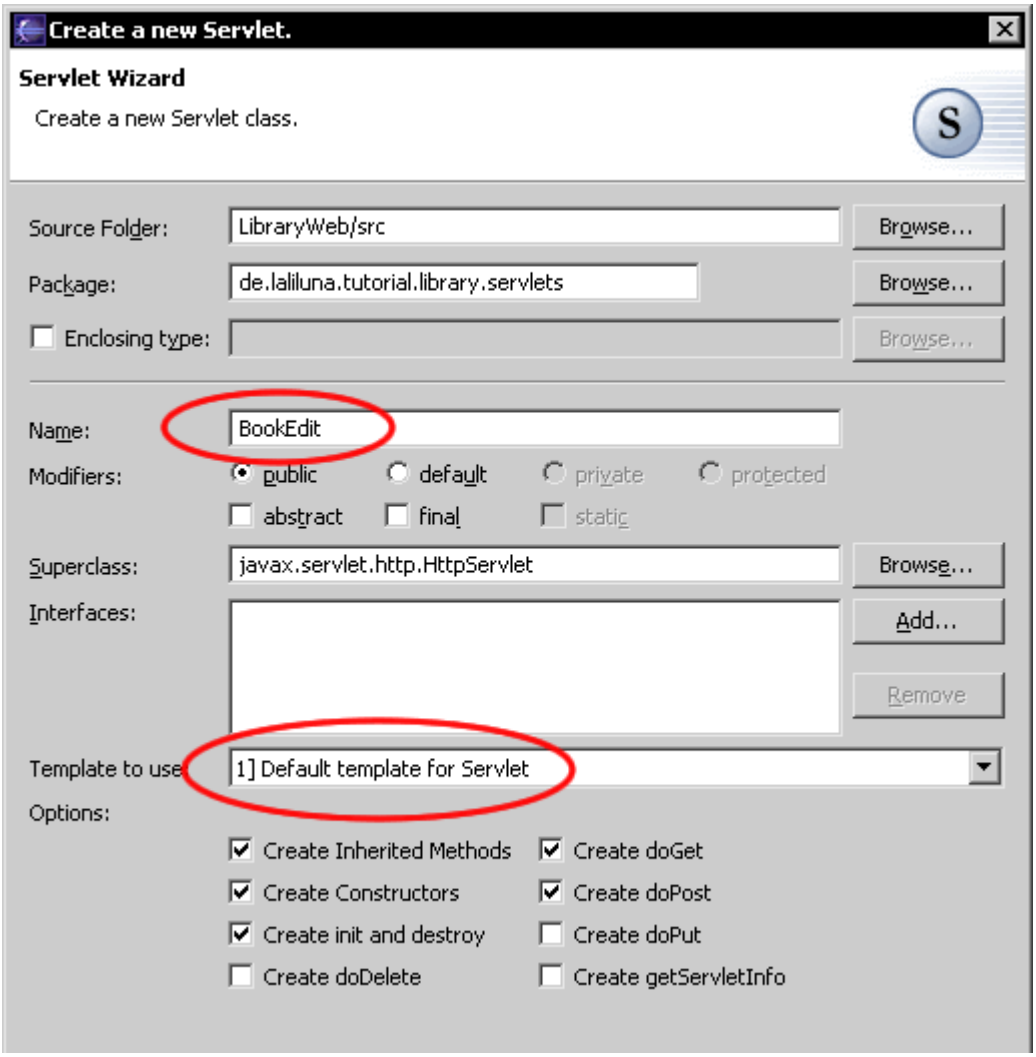
```
    }

    public void setAvailable(boolean available) {
        this.available = available;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

      public String getAuthor() {
            return author;
      }

      public void setAuthor(String author) {
            this.author = author;
      }

      public String getTitle() {
            return title;
      }

      public void setTitle(String title) {
            this.title = title;
      }
}
```

Create the servlets

In our example application we use two servlets, which are responsible for steering our web application.

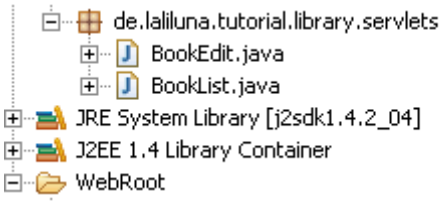
Create a new package `de.laliluna.tutorial.library.servlets` in `src` of your project.

Right click on the package and choose `New > Servlet`.



Repeat the steps for the second servlet. Set the name of the servlet to „BookList“.

Now you find the servlets in your package.



Edit the servlets

BookList.java

Open the servlet class `BookList.java`. This servlet loads a list of books and saves it in the request. Later in the JavaServer Page (jsp) we use the list to display the books.

Change the method `doGet(...)` like the following.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //hole die session des requests
            HttpSession session = request.getSession();

            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();

            //lies alle Bücher aus der Datenbank Klasse
            Collection collection = simulateDB.getAllBooks(session);

            //setze die zurück gegebene Liste der Bücher im Request
            request.setAttribute("books", collection);

            //hole den Request Dispatcher für die JSP Datei
            RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookList.jsp");

            //leite auf die JSP Datei zum Anzeigen der Liste weiter
            dispatcher.forward(request, response);
}
```

In the `doPost(...)` method we call the method `doGet(...)`, because we want to execute the same logic for each request GET or POST of the browser.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //rufe die doGet(...) Methode auf
            this.doGet(request, response);

}
```

We finished the servlet, that gets all books of the library.

BookEdit.java

In the next step we want to edit the servlet class `BookEdit.java`. Later it should control add, edit, delete of the books.

Open the class `BookEdit.java` and change the method `doGet(...)` like the following. With the parameter do in the method `doGet(...)` we check which action is called.

When creating a book, it refers to the display jsp.

When the value edit is assigned to the parameter do, the book is loaded and saved in the request with the name book.

When deleting is choosen, the book is deleted and the application is forwarded to the servlet BookList to display the list of books.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

            //hole die session des requests
            HttpSession session = request.getSession();
```

```
            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();

            //lies den Parameter do aus dem Request
            String action = request.getParameter("do");

            //lies den Parameter id aus dem Request
            long id = 0;
            try{
                  id = Long.parseLong(request.getParameter("id"));
            }catch(NumberFormatException e){}

            //hinzufügen eines Buches
            if(action.equals("add")){

                  //hole den Request Dispatcher für die JSP Datei
                  RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookAdd.jsp");

                  //leite auf die JSP Datei zum Anlegen eines Buches weiter
                  dispatcher.forward(request, response);

            //bearbeiten eines Buches
            }else if(action.equals("edit")){

                  //lies das Buch anhand seiner Id aus
                  Book book = simulateDB.loadBookById(id, session);

                  //setze das Buch Objekt im request
                  request.setAttribute("book", book);

                  //hole den Request Dispatcher für die JSP Datei
                  RequestDispatcher dispatcher = getServletContext().
getRequestDispatcher("/jsp/bookEdit.jsp");

                  //leite auf die JSP Datei zum Bearbeiten eines Buches weiter
                  dispatcher.forward(request, response);


            //löschen eines Buches
            }else if(action.equals("delete")){

                  //lösche das Buch anhand der Id
                  simulateDB.deleteBookById(id, session);

                  //Redirect Weiterleitung zum BookList Servlet
                  response.sendRedirect(request.getContextPath() + "/bookList");

            }

}
```

In this servlet we do **not** call the `doGet(...)` method within the method `doPost(...)`. We distinguish between the GET or POST request of the browser.

If the browser sends a post request to the servlet (send a form with <form method="post">) the transfered data, in our case the properties of the book (author, title, ...), will be processed.

Change the `doPost(...)` method to the following.
Within the method the transfered properties are read from the request and a new object book is created. The method saveToDB(...) of the database class saves the object book. After saving, the servlets it forwards to the servlet, that display the list of books.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

            //hole die session des requests
            HttpSession session = request.getSession();
```

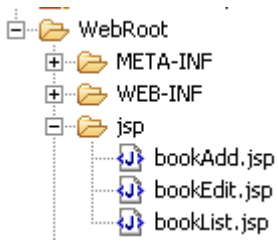
```
            //initialisiere die Datenbank Klasse
            SimulateDB simulateDB = new SimulateDB();

            //lies den Parameter id aus dem Request
            long id = 0;
            try{ id = Long.parseLong(request.getParameter("id")); }
            catch(NumberFormatException e){}

            //lies die Buch-Eigenschaften aus dem Request
            String author = request.getParameter("author");
            String title = request.getParameter("title");
            Boolean available = Boolean.valueOf(request.getParameter
("available"));

            //erstelle ein neues Buch Objekt
            Book book = new Book(id, author, title, available.booleanValue());

            //speichere das Buch Objekt
            simulateDB.saveToDB(book, session);

            //Redirect Weiterleitung zum BookList Servlet
            response.sendRedirect(request.getContextPath() + "/bookList");
}
```

Create the jsp files

To display the list of books and the dialogs to add and edit a book, we create three jsp files.

Create a new folder `jsp` in `/WebRoot` and add the three new JavaServer Pages.

- bookAdd.jsp
- bookEdit.jsp
- bookList.jsp



bookAdd.jsp

Open the first jsp file `bookAdd.jsp`. Later the JavaServer Page will display a form, where the user can add a book. This jsp file is only for adding new data and send this data to the servlet.

```
<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book add page</title>
  </head>
  <body>
      <form name="edit" action="bookEdit" method="post">
            <table border="1">
```

```
                  <tbody>
                        <tr>
                              <td>Author:</td>
                              <td><input type="text" name="author"
value=""></td>
                              </tr><tr>
                              <td>Title:</td>
                              <td><input type="text" name="title" value=""></td>
                              </tr><tr>
                              <td>Available:</td>
                              <td><input type="checkbox" name="available"
value="true"></td>
                              </tr><tr>
                              <td colspan="2"><input type="submit"
name="btnSave" value="Save"></td>
                        </tr>
                  </tbody>
            </table>
      </form>
  </body>
</html>
```

bookEdit.jsp

The JavaServer Page `bookEdit.jsp` reads the object book from the request and set the properties to the values of the form fields. In the first line the package de.laliluna.tutorial.library and all subordinated packages are imported to the jsp file. It is called a jsp-directive. This directive is similar to the import command within a java class.

The source code looks like the following.

```
<%@ page import="de.laliluna.tutorial.library.*" %>
<%
      //lies das Objekt book aus dem Request
      Book book = (Book)request.getAttribute("book");
%>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book edit page</title>
  </head>
  <body>
      <form name="edit" action="bookEdit" method="post">
            <table border="1">
                  <tbody>
                        <tr>
                              <td>Author:</td>
                              <td><input type="text" name="author" value="<%
=book.getAuthor()%>"></td>
                              </tr><tr>
                              <td>Title:</td>
                              <td><input type="text" name="title" value="<%
=book.getTitle()%>"></td>
                              </tr><tr>
                              <td>Available:</td>
                              <td><input type="checkbox" name="available"
value="true" <% if(book.isAvailable()) out.println("checked"); %>></td>
                              </tr><tr>
```

```
                              <td colspan="2"><input type="submit"
name="btnSave" value="Save"></td>
                        </tr>
                  </tbody>
            </table>
            <input type="hidden" name="id" value="<%=book.getId()%>">
      </form>
  </body>
</html>
```

bookList.jsp

The jsp file `bookList.jsp` displays the list of books, saved by the servlet. An array list of books is read from the request and looped over. Within the `for` loop we print out the properties of the books with the method `println(..)`.

```
<%@ page language="java" import="java.util.*" %>
<%@ page import="de.laliluna.tutorial.library.*" %>

<%
String path = request.getContextPath();
String basePath = request.getScheme()+"://"+request.getServerName()
+":"+request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <base href="<%=basePath%>">

    <title>Book list page</title>
  </head>
  <body>
      <table border="1">
            <tbody>
            <tr>
                  <td>Author</td>
                  <td>Book name</td>
                  <td>Available</td>
                  <td> </td>
                  <td> </td>
            </tr>
        <%
            //lies die Liste der Bücher aus dem Request
            ArrayList arrayList = (ArrayList)request.getAttribute("books");

            //loope über die Liste von Büchern und gibt die Eigenschaften aus
            for (Iterator iter = arrayList.iterator(); iter.hasNext();) {
                  Book element = (Book) iter.next();

                  out.println("<tr>");
                  out.println("<td>" + element.getAuthor() + "</td>");
                  out.println("<td>" + element.getTitle() + "</td>");
                  if(element.isAvailable())
                        out.println("<td><input type=\"checkbox\"
name=\"available\" value=\"true\" disabled checked></td>");
                  else
                        out.println("<td><input type=\"checkbox\"
name=\"available\" value=\"true\" disabled></td>");

                  out.println("<td><a href=\"bookEdit?do=edit&id=" +
element.getId() + "\">Edit</a></td>");
                  out.println("<td><a href=\"bookEdit?do=delete&id=" +
element.getId() + "\">Delete</a></td>");
                  out.println("</tr>");
            }
        %>
```

```
            </tbody>
        </table>
        <br>
        <a href="bookEdit?do=add">Add a new book</a>
    </body>
</html>
```

We have finished the JavaServer pages to display the data.

Default JSP

When the user call the project in the browser with http://localhost:8080/LibraryWeb/ , we want to redirect him to a welcome page. Create an `index.jsp` file in `/WebRoot`. This jsp file redirects to the page, which displays the list of books.

Change the source code to the following.

```
<%
        //redirect to the book list servlet
        response.sendRedirect("bookList");
%>
```

Edit the web.xml

In the next step, we edit the configuration file of the project, the web.xml.

Open the file web.xml and add the following configuration.

The tag `<servlet-name>` assigns a name to our servlet class. With this name we can identify the servlet in the `web.xml`. `<servlet-class>` define the class for this name. Within the tag `<servlet-mapping>` we assign a address (URL) to a servlet, so a user can later execute this servlet by calling this address in his browser.

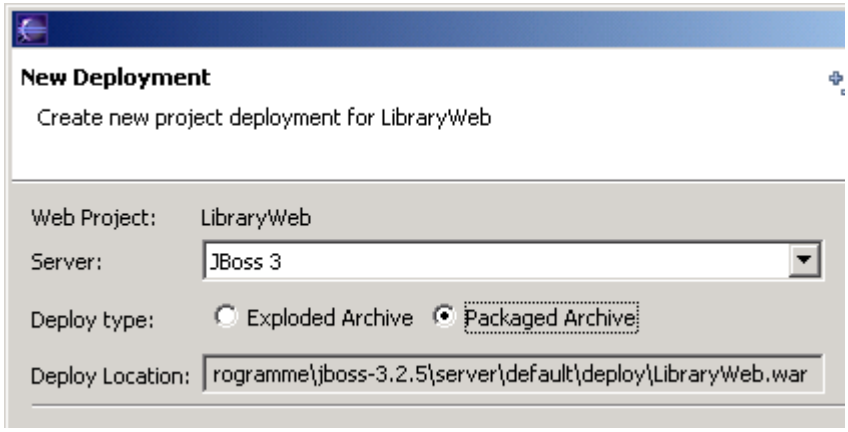
The servlets are called with /bookList and /bookEdit.

```
<web-app>
<servlet>
        <servlet-name>BookList</servlet-name>
        <servlet-class>de.laliluna.tutorial.library.servlets.BookList</servlet-
class>
</servlet>
<servlet-mapping>
        <servlet-name>BookList</servlet-name>
        <url-pattern>/bookList</url-pattern>
</servlet-mapping>

<servlet>
        <servlet-name>BookEdit</servlet-name>
        <servlet-class>de.laliluna.tutorial.library.servlets.BookEdit</servlet-
class>
</servlet>
<servlet-mapping>
        <servlet-name>BookEdit</servlet-name>
        <url-pattern>/bookEdit</url-pattern>
</servlet-mapping>
</web-app>
```

Thats all.

Test the application

Deploy the project to the jboss.



Test your application http://localhost:8080/LibraryWeb