First steps using Struts and Hibernate

In this tutorial we will show how the Web Framework Struts and the Database Persistence Solution Hibernate can be used together. Though we explain some basic, you should try the introductionary tutorials first when you are a beginner.

General

Author:
Sebastian Hennebrüder

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

Date:
Third Revision, July, 27th 2005
Revised January, 16th 2005
First Edition December, 22th 2004

Source code:

http://www.laliluna.de/assets/tutorials/struts-hibernate-integration-tutorial.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.

PDF Version of the Tutorials:

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

Development Tools
Eclipse 3.x
Hibernate 2
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.)
Database
PostgreSQL 8.0 or MySQL
Application Server
Jboss 4.0.3
You may use Tomcat here if you like.

Table of Content

First steps using Struts and Hibernate..... 1
General..... 1
Requirements..... 2
Creating the application..... 2
Create the project and add the Hibernate capabilities..... 2
Notice for non MyEclipse users..... 5
Reduce Hibernate Libraries..... 6
Create the Database..... 7

Generate the Hibernate Mapping Files and Classes..... 7
Repair the Hibernate mapping..... 10
Correct the Boolean mapping..... 11
Improvements to the session factory..... 12
Testing the Hibernate part..... 12
PostgreSQL Problem..... 17
Generating the Business Logic..... 18
Create a business logic class..... 18
Creating the dialogs with Struts..... 25
Create a default, welcome page..... 26
Global Action Forwards and Action Mappings..... 28
Book list..... 31
Action mapping und action class of the book list..... 32
Edit the source code of the action form class..... 33
Edit the source code of the action class..... 33
Display the books list in the jsp file..... 34
Test the application..... 36
Add, edit, borrow and delete books..... 36
Action Mapping 36
Edit the source code of the jsp files..... 39
Form bean 41
Methods of the dispatch action class..... 44
Use case Customer list..... 47
Edit the source code of the action form class..... 49
Displaying the custom list..... 50
Use case add, edit, delete customers..... 51
Customer form bean..... 54
Edit the source code of the action class..... 56
Edit the source code of the jsp file..... 57
Test the applications..... 58

Requirements

We will use the IDE Eclipse with the plugin MyEclipse in this tutorial. But you are not forced to use it, as we will explain what the MyEclipse wizards created actually. Have a look at the colored notice .

You may try MyEclipse, as it is not expensive. There is also a trial version available:
http://www.lalliluna.de/myeclipse.html

If you want to use free tools for web application development, have a look at the tutorial

http://www.lalliluna.de/first-steps-with-struts-free-tools-en.html

Creating the application

We will start with creating and testing of the persistence layer. The second step is to add the business logic and at last will integrate the Struts part.

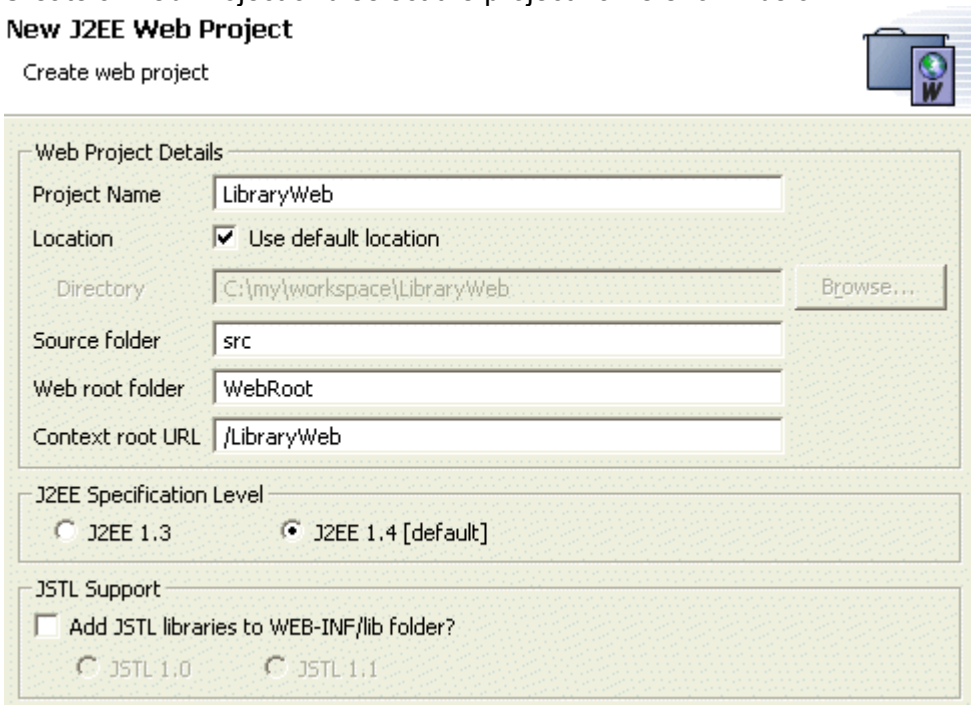
Create the project and add the Hibernate capabilities

Create a new web project.

So let's start.

Press „Strg + N“ to open the „New ...“ dialog.

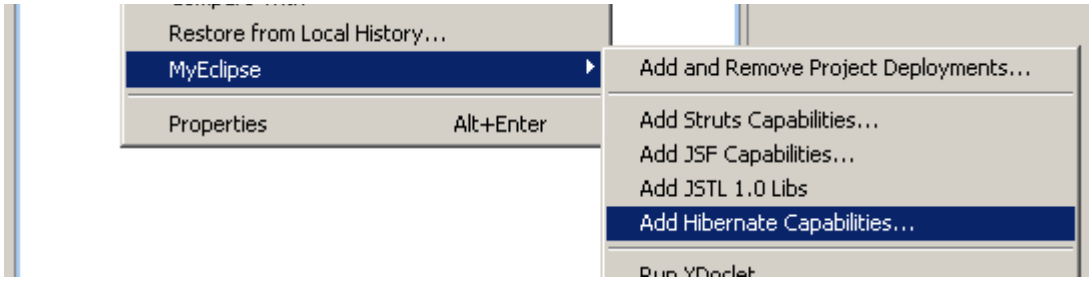
Create a Web Project and select the project name shown below.



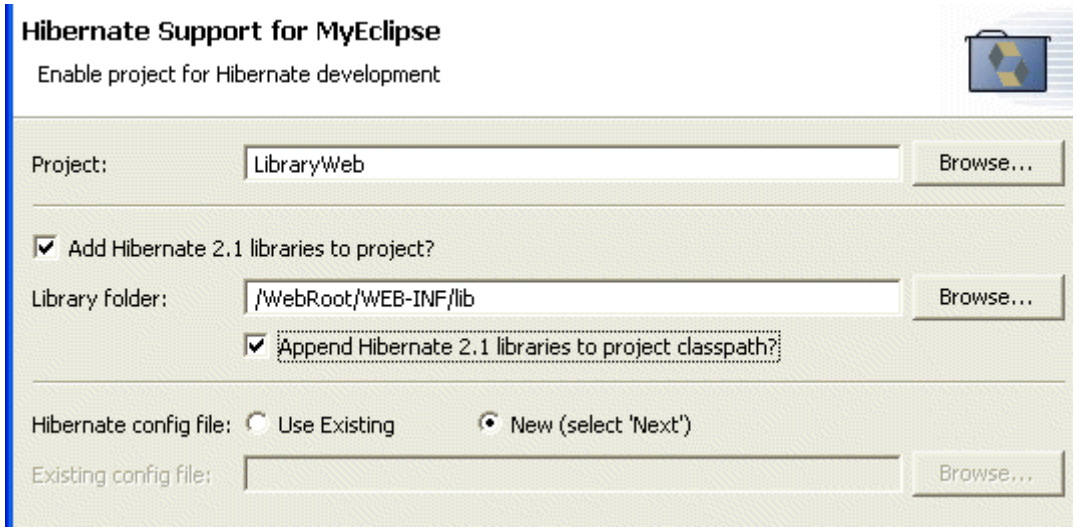
Notice

MyEclipse provides functionality to create and deploy web projects to a wide choice of application server. You may use Ant to do this by hand or have a look in the tutorial stated above to create web projects with free tools.

Add the Hibernate capabilities by right clicking on the project in the Package View.



Check the two checkboxes to add the libraries to the project and select to create a new hibernate mapping file. The hibernate file holds the configuration of your hibernate settings and mappings.



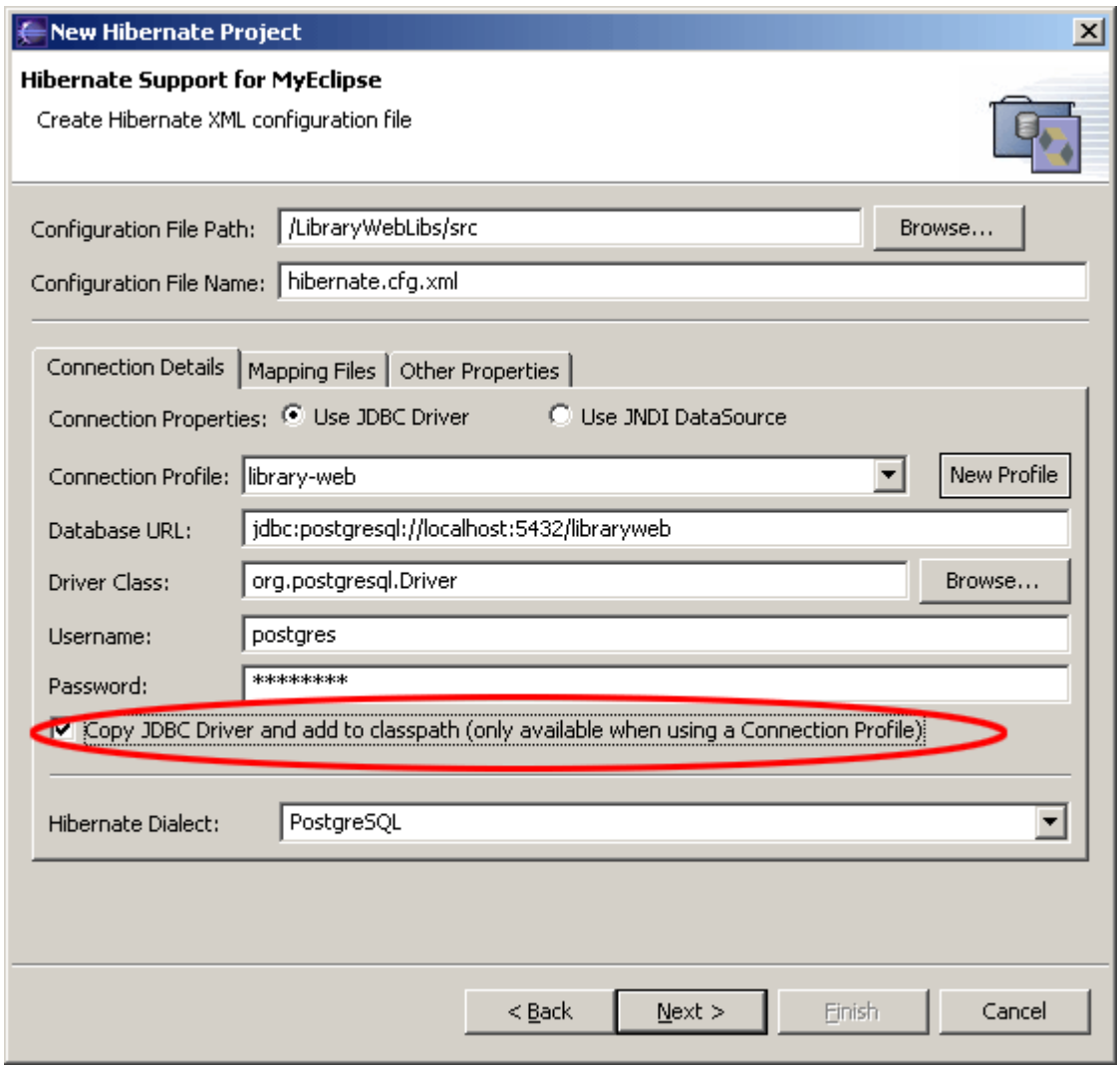
The next step is to select a Connection Profile for the Database.

Select the button „New profile“ to create a new profile.

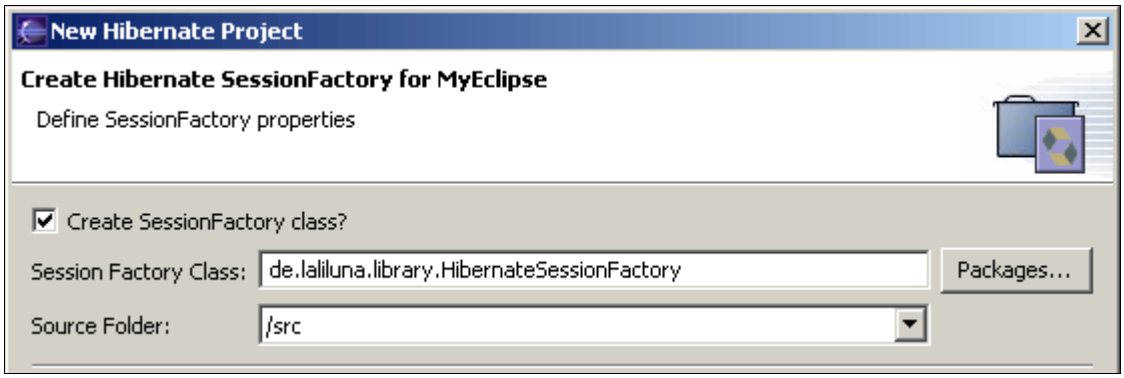
When the Postgre Driver is missing. Click on „New Driver“ to create a new driver. You will need the jar including your Database Driver.

We call our profile library-web. Specify the user name and the password and create it.

Back to the first dialog, make sure that you have the checkbox „Copy JDBC Driver ...“ selected. We are going to use PostgreSQL. It should not be difficult to make the same thing for MySQL or another database. Make sure that you have the Jar File of the Database Driver somewhere on your disc.



In the next step you must invent a nice name for your SessionFactory.



What is a SessionFactory?

A session factory creates a Hibernate session for you, so it is not very complicated.

Hibernate expects that only one instance of the Hibernate Session Class is used per thread. Normally you would have to create a class implementing a ThreadLocal. MyEclipse does this for you. Your only have the difficult part to invent a name for it. If you are not using MyEclipse have a look in the sources.

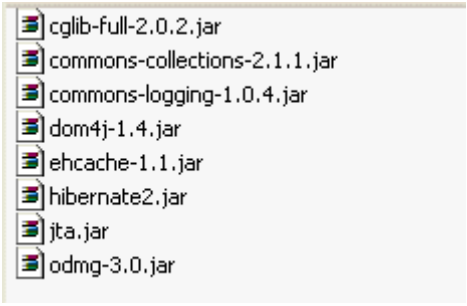
Notice for non MyEclipse users

Notice for non MyEclipse users

What the wizards did is to add all the libraries used by Hibernate, create the hibernate configuration file and create a SessionFactory. You can do this easily by hand.

Download Hibernate from http://www.hibernate.org/

As minimum requirement add the following libraries to get Hibernate 2 to work. For Hibernate 3 please verify if there are additional libraries needed. When you want to know more about the libraries and if they are required, have a look at the file README.txt included in the lib directory of the hibernate.zip.



The configuration file is a simple XML file named

hibernate.cfg.xml

In our case we will put it directly in the src directory. Create an XML file there and add the following content.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 2.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<!-- DO NOT EDIT: This is a generated file that is synchronized -->
<!-- by MyEclipse Hibernate tool integration.                   -->
<hibernate-configuration>

    <session-factory>
        <!-- properties -->
        <property name="connection.username">postgres</property>
        <property
name="connection.url">jdbc:postgresql://localhost:5432/libraryweb</property>
        <property
name="dialect">net.sf.hibernate.dialect.PostgreSQLDialect</property>
        <property name="connection.password">p</property>
        <property
name="connection.driver_class">org.postgresql.Driver</property>

        <!-- mapping files -->
        <!-- The following mapping element was auto-generated in -->
        <!-- order for this file to conform to the Hibernate DTD -->
        <mapping resource="Add valid path"/>

    </session-factory>

</hibernate-configuration>
```

Then create the **HibernateSessionFactory** class in the package **de.lalliluna.library** and add the content as included in the sources with this tutorial.

That's all for the non MyEclipse users.

Reduce Hibernate Libraries

By default MyEclipse includes a heavy load of libraries. Some of them will only be needed for local development others only for special cache implementations. When you want to optimize your

deployment after you learned the basics of Hibernate download Hibernate from the website http://www.hibernate.org/ In the lib directory you will find a README.txt explaining what libraries are optional.

Now we are prepared to start the development. Fasten the seatbelts, it is getting really fast now.

Create the Database

Create the database and the following tables. Do not forget the foreign key!

Postgre SQL Script

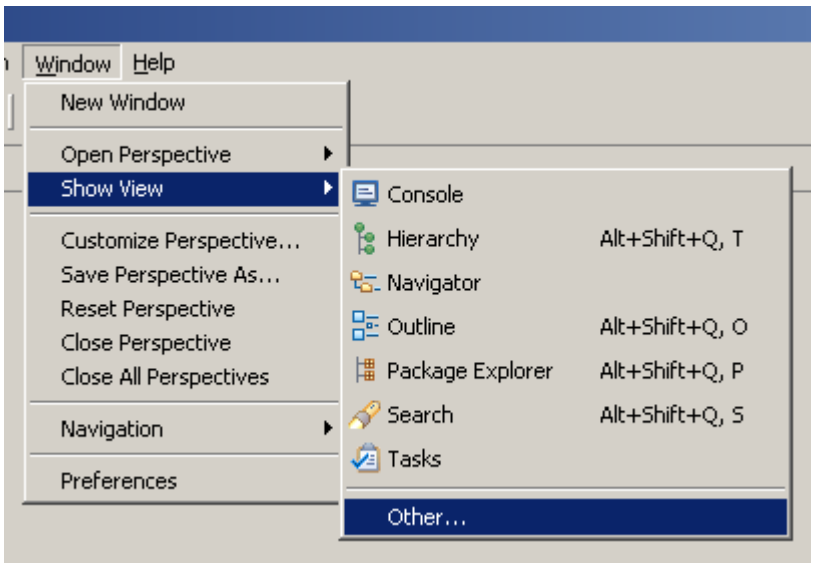
```
CREATE TABLE customer
(
  id serial NOT NULL,
  name text,
  lastname text,
  age int4,
  CONSTRAINT customer_pk PRIMARY KEY (id)
) ;
CREATE TABLE book
(
  id serial NOT NULL,
  title text,
  author text,
  customer_fk int4,
  available bool,
  CONSTRAINT book_pk PRIMARY KEY (id)
) ;
ALTER TABLE book
  ADD CONSTRAINT book_customer FOREIGN KEY (customer_fk) REFERENCES customer
(id) ON UPDATE RESTRICT ON DELETE RESTRICT;
```

MySQL Script

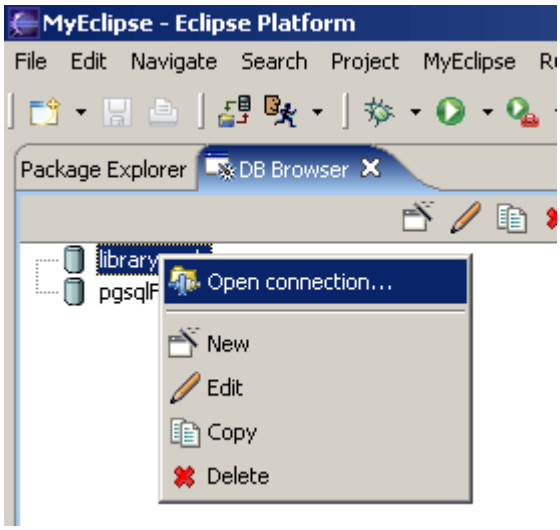
```
CREATE TABLE customer
(
  id int( 11  )  NOT  NULL  AUTO_INCREMENT ,
  name varchar( 255  ) ,
  lastname  varchar( 255  ) ,
  age int( 11  ),
  CONSTRAINT customer_pk PRIMARY KEY (id)
) TYPE=INNODB;
CREATE  TABLE book( id  int( 11  )  NOT  NULL  AUTO_INCREMENT ,
title varchar( 255  ) ,
author varchar( 255  ) ,
customer_fk int( 11  ),
available TINYINT NOT NULL,
  CONSTRAINT book_pk PRIMARY  KEY ( id ),
INDEX (customer_fk)  ) TYPE=INNODB;
 ALTER  TABLE book ADD  CONSTRAINT book_customer FOREIGN  KEY ( customer_fk )
REFERENCES customer( id )  ON  UPDATE  RESTRICT  ON  DELETE  RESTRICT ;
```

Generate the Hibernate Mapping Files and Classes

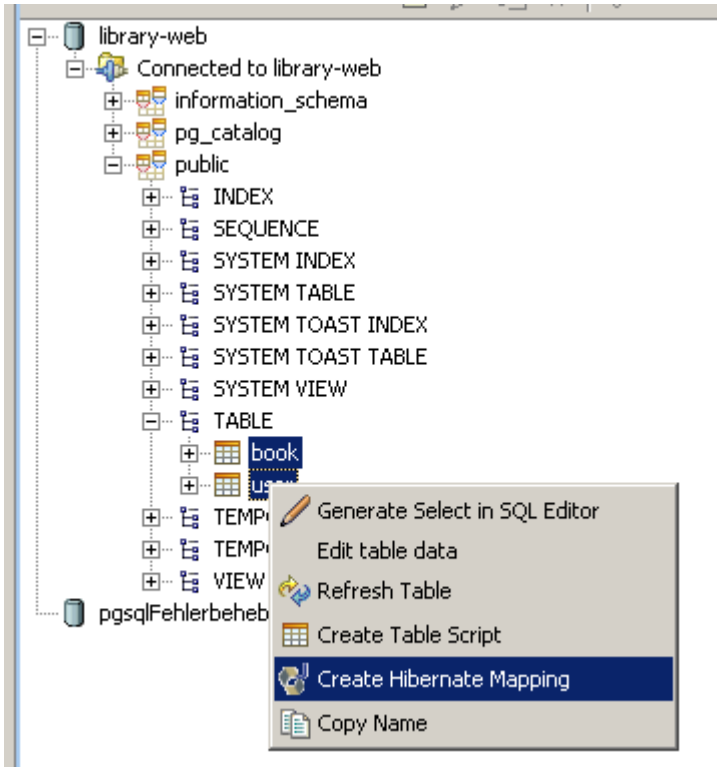
Open the View DB Browser (MyEclipse). If you cannot find it open the „Show View“ Dialog and select in the MyEclipse Enterprise Workbench the DB Browser.



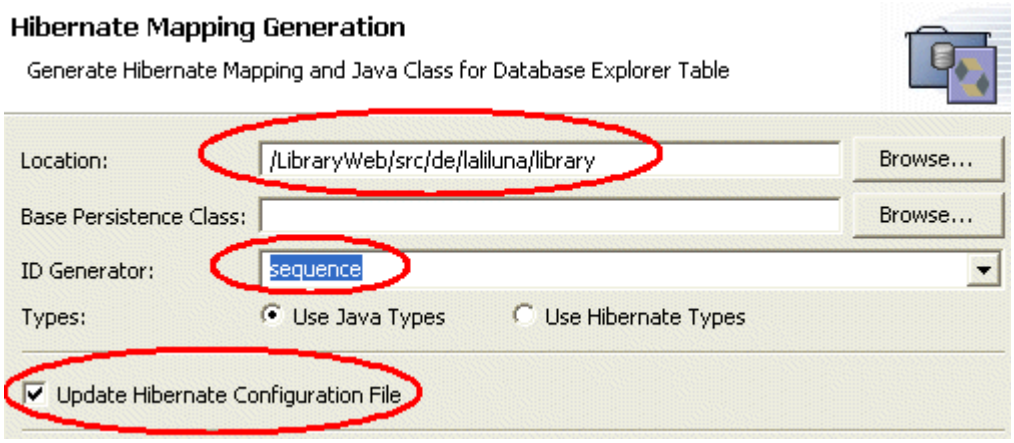
Open the connection profile you specified before.



Select the two tables we have just created. Right click and choose „Create Hibernate Mapping“.



Select your LibraryWeb project as target. When your are using Postgre select „sequence“ as ID Generator. When you are using MySQL select „native“.



Click OK and your are really good! You have just created your persistence layer ;-)

Now we will have a closer look at our package explorer to see what happened.

First open the hibernate.cfg.xml.

There are two new entries, specifying where the two mapping files are located. It is a good idea to keep the mapping files separated from the **hibernate.cfg.xml**. (What MyEclipse actually does for you.)

```
        <!-- mapping files -->
        <mapping resource="de/laliluna/library/Book.hbm.xml"/>
        <mapping resource="de/laliluna/library/Customer.hbm.xml"/>
```

Have a look at the mapping file Book.hbm.xml. In this file the mapping from the class and its attributes to the table fields is specified. Even our foreign key has been recognized.

```
<hibernate-mapping package="de.laliluna.library">
    <class name="Book" table="book">
        <id name="id" column="id" type="java.lang.Integer">
            <generator class="sequence"/>
        </id>

        <property name="title" column="title" type="java.lang.String" />
        <property name="author" column="author" type="java.lang.String" />
        <property name="available" column="available" type="java.lang.Byte" />
        <many-to-one name="customer" column="customer_fk" class="Customer" />
    </class>
</hibernate-mapping>
```

When you are using MySQL the mapping is slightly different.

```
   <class name="Book" table="book">
        <id name="id" column="id" type="java.lang.Integer">
            <generator class="native"/>
        </id>
...
```

MyEclipse created two files per class. The first one is an abstract class. (AbstractBook) It will be overwritten each time you repeat the import procedure. In the second class (Book) you may adapt any changes you want to make. It is only generated once.

Notice

Non MyEclipse users please take the files Book.hbm.xml, AbstractBook, Book, customer.hbm.xml, AbstractCustomer and Customer from the sources provided with this tutorial.

Hibernate does also provide tools to create mapping files. Have a look at the hibernate website.

Repair the Hibernate mapping

We are going to make some changes.

Hibernate do not generate a relation back from the customer to the book. We will add this by hand.

In the file Customer.class add the following.

```
    private List books;
    /**
     * @return Returns the books.
     */
    public List getBooks() {
      return books;
    }
    /**
     * @param books The books to set.
     */
    public void setBooks(List books) {
      this.books = books;
    }
```

In the file Customer.hbm.xml we have to add the mapping from the books variable. Add the "bag" entry to the file.

```
<hibernate-mapping package="de.laliluna.library">
    <class name="Customer" table="customer">
        <id name="id" column="id" type="java.lang.Integer">
```

```
            <generator class="sequence"/>
        </id>

        <bag name="books" inverse="false">
        <key column="customer_fk" />
        <one-to-many class="Book"/>
        </bag>

        <property name="name" column="name" type="java.lang.String" />
        <property name="lastname" column="lastname" type="java.lang.String" />
        <property name="age" column="age" type="java.lang.Integer" />
    </class>

</hibernate-mapping>
```

We specify it as

```
inverse="false"
```

This specifies that we want changes to the attribute on the "one side" of a one to many relation (books) to be reflected in the database.

For example:

customer.getbooks().add(aBook);

should write the foreign key to the customer table.

Manually changing the file above is not very good but there is no other way here.

The disadvantage is that this will be overwritten each time you regenerate the mapping files. In our case it is not so important but in a larger project this will make it impossible to use the autogeneration from MyEclipse except at the beginning. The hibernate import function is quite new for MyEclipse, so you can be sure that there will be larger improvements in the next versions.

Notice

I had a problem copy and pasting source code from the tutorial to Eclipse XML files due to use of tabs in this document. You may make the changes by hand if you encounter any problems.

Correct the Boolean mapping

The postgre bool column is recognized as Byte and as Short when you are using MySql. Do not ask me why.

This is not very nice, so we will correct this.

Change the Hibernate mapping in the file Book.hmb.xml to

```
<property name="available" column="available" type="java.lang.Boolean" />
```

Change the variable and the getter and setter in the file AbstractBook.java to Boolean type. private

```
java.lang.Boolean available;
    /**
     * Return the value of the available column.
     * @return java.lang.Byte
     */
    public java.lang.Boolean getAvailable()
    {
        return this.available;
    }

    /**
     * Set the value of the available column.
     * @param available
     */
    public void setAvailable(java.lang.Boolean available)
    {
```

```
        this.available = available;
    }
```

Improvements to the session factory

The session factory generated by MyEclipse is not very nice because it lets you run into errors when you use the session.close() method. The session factory expects that you use the static method closeSession() of the factory, which actually sets the session to null if it is closed.

But no problem, here are the changes to the method currentSession of the factory.

```
  public static Session currentSession() throws HibernateException {
    Session session = (Session) threadLocal.get();
    /*
     * [laliluna] 20.12.2004
     * we want to use the standard session.close() method and not the
closeSession() from this class.
     * For this we need the following line of code.
     */
    if (session != null && !session.isOpen()) session = null;
    if (session == null) {
      if (sessionFactory == null) {
        try {
          cfg.configure(CONFIG_FILE_LOCATION);
          sessionFactory = cfg.buildSessionFactory();
        } catch (Exception e) {
          System.err
              .println("%%%% Error Creating HibernateSessionFactory %%%%");
          e.printStackTrace();
        }
      }
      session = sessionFactory.openSession();
      threadLocal.set(session);
    }
    return session;
  }
```

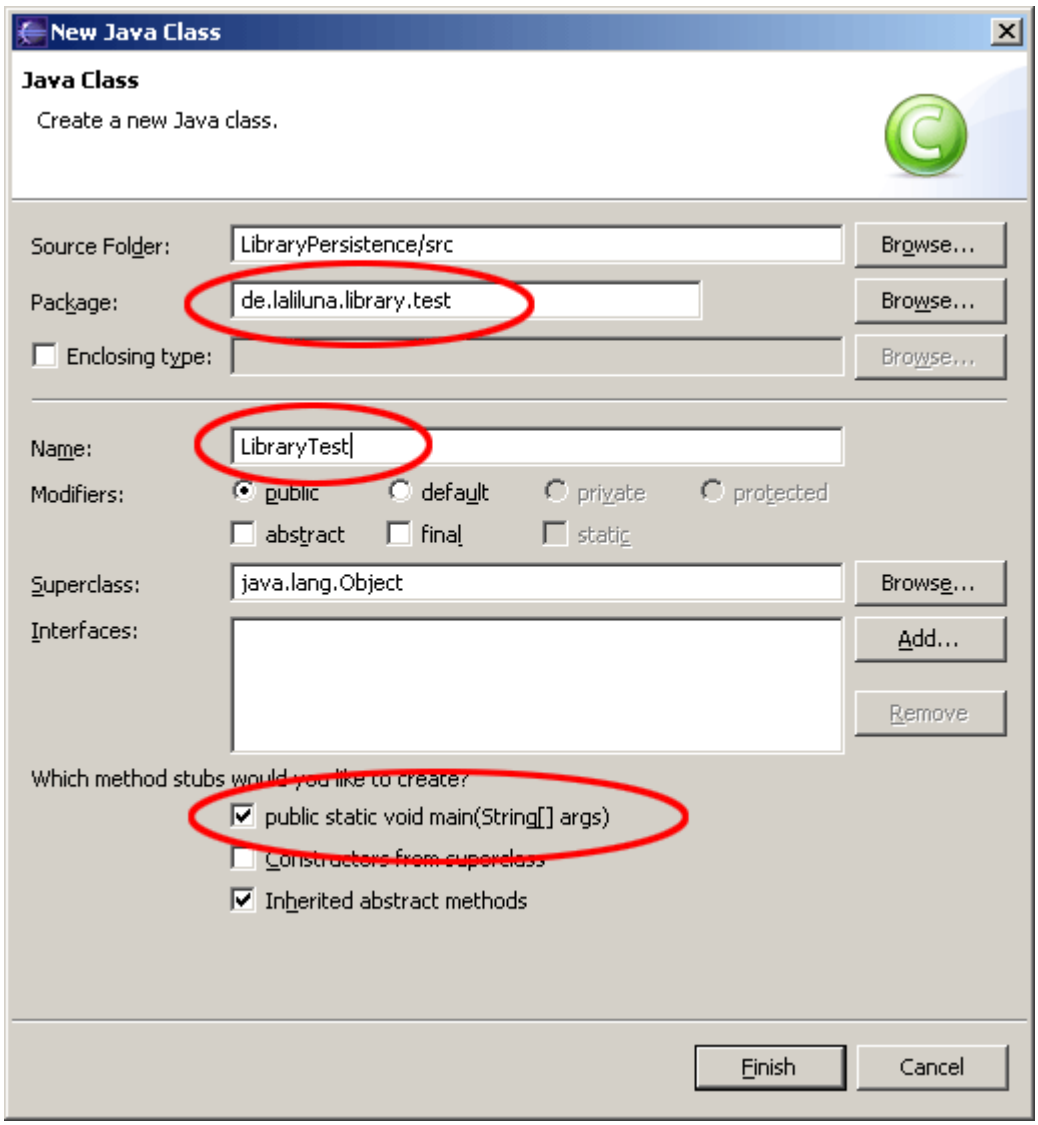
Testing the Hibernate part

We will need log4j to test Hibernate outside of an Application Server. You can find the library as jar file here:

http://logging.apache.org/

Add the library to your Eclipse project (project properties => Java Build Path => Add Jar or Add external Jar

Next step is to create a new class.



Add the following content:

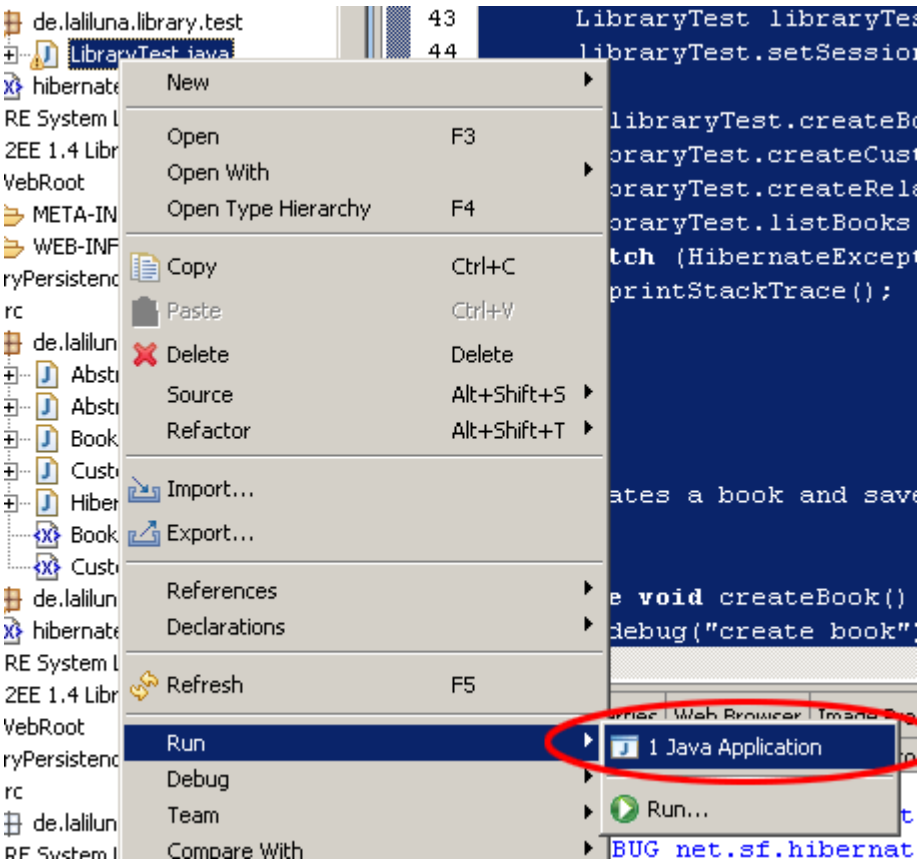
```
package de.laliluna.library.test;

import java.util.Iterator;
import java.util.List;

import org.apache.log4j.BasicConfigurator;

import de.laliluna.library.Book;
import de.laliluna.library.Customer;
import de.laliluna.library.HibernateSessionFactory;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Query;
import net.sf.hibernate.Session;
import net.sf.hibernate.Transaction;

public class LibraryTest {
  private Session session;
  private Logger log = Logger.getLogger(this.getClass());
  public static void main(String[] args) {
    /*
     * hibernate needs log4j. Either specify a log4j.properties file
     *
     *
PropertyConfigurator.configure
```

```
("D:\\_projekte\\workspace\\LibraryWeb\\src\\log4j.properties");
     *
     * or alternatively make the following to create a standard
configuration
     * BasicConfigurator.configure();
     */
    BasicConfigurator.configure();
    try {
      LibraryTest libraryTest = new LibraryTest();
      libraryTest.setSession(HibernateSessionFactory.currentSession());
      libraryTest.createBook();
      libraryTest.createCustomer();
      libraryTest.createRelation();
      libraryTest.deleteCustomer();
      libraryTest.listBooks();
      // [lalilуna] 20.12.2004 always close the session at the end
      libraryTest.getSession().close();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
  /**
   * creates a book and saves it to the db.
   *
   */
  private void createBook() {
    System.out.println("############## create book");
    try {
      Transaction tx = session.beginTransaction();
      Book book = new Book();
      book.setAuthor("Karl");
      book.setTitle("Karls biography");
      session.save(book);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
  /**
   * creates a customer and saves it to the db
   *
   */
  private void createCustomer() {
    System.out.println("############## create user");
    try {
      Transaction tx = session.beginTransaction();
      Customer customer = new Customer();
      customer.setLastname("Fitz");
      customer.setName("John");
      customer.setAge(new Integer(25));
      session.save(customer);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
  /**
   * creates a book and a customer  + a relation between the two
   *
   */
  private void createRelation() {
    System.out.println("############## create relation");
    try {
      Transaction tx = session.beginTransaction();
      Customer customer = new Customer();
      customer.setLastname("Schmidt");
      customer.setName("Jim");
      customer.setAge(new Integer(25));
      /* IMPORTANT You must save the customer first, before you can
```

```
assign him to the book.
       * Hibernate creates and reads the ID only when you save the
entry.
       * The ID is needed as it is the foreign key
       */
      session.save(customer);
      Book book = new Book();
      book.setAuthor("Gerhard Petter");
      book.setTitle("Gerhards biography");
      session.save(book);
      Book book2 = new Book();
      book2.setAuthor("Karl May");
      book2.setTitle("Wildes Kurdistan");
      session.save(book2);
      session.flush();
      book.setCustomer(customer);
      book2.setCustomer(customer);
      tx.commit();
      // [laliluna] 20.12.2004 the customer is not updated
automatically, so we have to refresh him
      session.refresh(customer);
      tx = session.beginTransaction();
      if (customer.getBooks() != null) {
        System.out.println("list books");
        for (Iterator iter = customer.getBooks().iterator();
iter.hasNext();) {
          Book element = (Book) iter.next();
          System.out.println("customer:" + element.getCustomer());
          System.out.println("customer is now:" +
element.getCustomer());
        }
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
  private void deleteCustomer() {
    System.out.println("############## delete customer");
    try {
      Transaction tx = session.beginTransaction();
      Customer customer = new Customer();
      customer.setLastname("Wumski");
      customer.setName("Gerhard");
      customer.setAge(new Integer(25));
      /* IMPORTANT You must save the customer first, before you can
assign him to the book.
       * Hibernate creates and reads the ID only when you save the
entry.
       * The ID is needed as it is the foreign key
       */
      session.save(customer);
      Book book = new Book();
      book.setAuthor("Tim Tom");
      book.setTitle("My new biography");
      session.save(book);
      book.setCustomer(customer);
      tx.commit();
      // [laliluna] 20.12.2004 and now we are going to delete the
customer which will set the foreign key in the book table to null
      tx = session.beginTransaction();
      //    [laliluna] 20.12.2004 the customer is not updated
automatically, so we have to refresh him
      session.refresh(customer);
      session.delete(customer);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
```

```
  }
  /**
   * lists all books in the db
   *
   */
  private void listBooks() {
    System.out.println("####### list customers");
    Query query;
    Transaction tx;
    try {
      tx = session.beginTransaction();
      query = session.createQuery("select c from Customer as c");
      for (Iterator iter = query.iterate(); iter.hasNext();) {
        Customer element = (Customer) iter.next();
        List list = element.getBooks();
        System.out.println(element.getName());
        if (list == null)
          System.out.println("list = null");
        else {
          for (Iterator iterator = list.iterator();
iterator.hasNext();) {
            Book book = (Book) iterator.next();
            System.out.println(book.getAuthor());
          }
        }
        System.out.println(element);
      }
      tx.commit();
    } catch (HibernateException e1) {
      e1.printStackTrace();
    }
    System.out.println("####### list books");
    try {
      tx = session.beginTransaction();
      query = session.createQuery("select b from Book as b");
      for (Iterator iter = query.iterate(); iter.hasNext();) {
        System.out.println((Book) iter.next());
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
  /**
   * @return Returns the session.
   */
  public Session getSession() {
    return session;
  }
  /**
   * @param session The session to set.
   */
  public void setSession(Session session) {
    this.session = session;
  }
}
```

Right click on the class and choose Run -> Java Application.



And at least when we are using PostgreSQL, we got a lots of error message. ;-)

java.sql.SQLException: ERROR: relation "hibernate_sequence" does not exist

PostgreSQL Problem

This is because there is a simple bug in the import script. It assumes that the sequence is called hibernate_sequence. The sequences created automatically when your are using a serial column, is called table_column_seq, eg: book_id_seq.

The easiest work around is to wait until MyEclipse improves the script. **The fastest** to create a sequence called hibernate_sequence. A disadvantage is that all tables share the same sequence. You will have one table under heavy load.

```
CREATE SEQUENCE  hibernate_sequence
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
```

The nicest way, but only possible when you are sure not to regenerate your mapping files (you would override your changes) is to change the mapping from

```
<generator class="sequence"/>
```

to the following for the book. The changes for the customer are analogues.

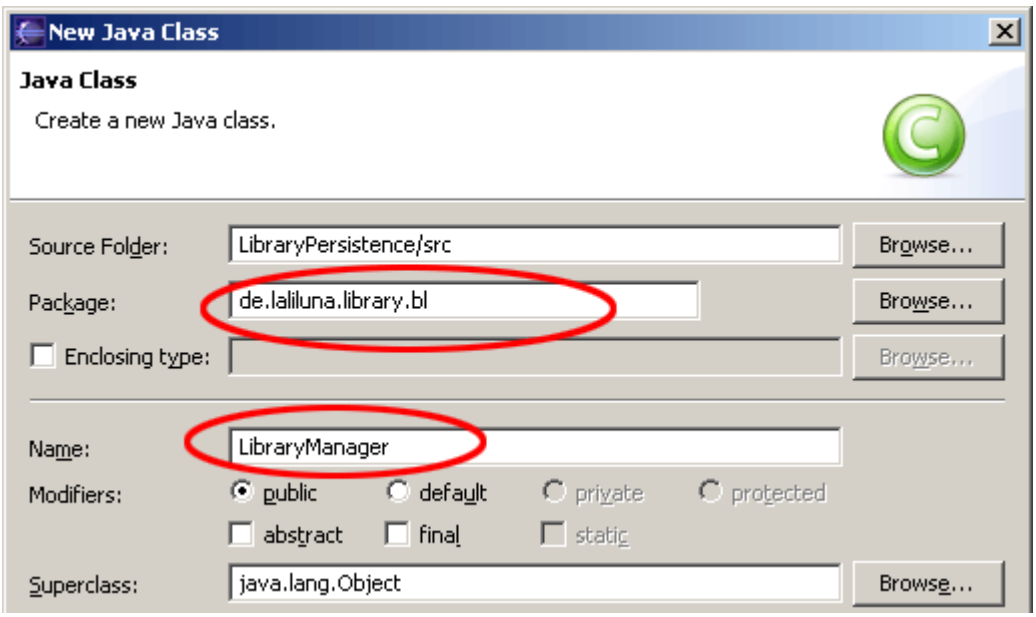
```
<generator class="sequence">
<param name="sequence">book_id_seq</param>
</generator>
```

That's it for the persistence layer for our application.

Generating the Business Logic

Create a business logic class

We will place all business logic in a single class. Later our Struts part will only use this Class. There won't be any direct access to the persistence layer. You could even think about replacing your persistence layer with another one.



This class will hold all methods we need as business logic

- creating, updating and deleting books
- creating, updating and deleting customers
- borrowing and returning books
- reading all customers or books from the db into a list

Hibernate exceptions

When an exceptions occurs it is recommended to roll back the transaction and to immediately close the session. That is what we have done with the

try

catch {}

finally{}

Hibernate Design we used

A hibernate query returns a List interface to a special Hibernate implementation of a List. This implementation is directly connected to the session. You cannot close your session when you use this Hibernate lists. Either you have to disconnect the session from the database and reconnect it, use one of the caching solutions or take the easiest but not best way to work with Value Objects.

We took the easiest way:

The consequence is that we have to copy all elements of a hibernate list to a normal java.util.List.

```
/*
```

```
 * Created on 25.11.2004 by HS
 *
 */
package de.laliluna.library.bl;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import de.laliluna.library.Book;
import de.laliluna.library.Customer;
import de.laliluna.library.HibernateSessionFactory;
import net.sf.hibernate.HibernateException;
import net.sf.hibernate.Query;
import net.sf.hibernate.Session;
import net.sf.hibernate.Transaction;

/**
 * @author HS
 *
 *
 */
public class LibraryManager {

  /**
   * get all books from the database
   * @return Array of BookValue
   */
  public Book[] getAllBooks() {
    /* will hold the books we are going to return later */
    List books = new ArrayList();
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;
    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Query query = session
          .createQuery("select b from Book as b order by b.author, b.title");
      for (Iterator iter = query.iterate(); iter.hasNext();) {
        books.add((Book) iter.next());
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }

    }
    return (Book[]) books.toArray(new Book[0]);
  }

  /**
   * get book by primary key
   * @param primaryKey
```

```
   * @return a Book or null
   */
  public Book getBookByPrimaryKey(Integer primaryKey) {
    /* holds our return value */
    Book book = null;
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      book = (Book) session.get(Book.class, primaryKey);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [lalilуna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();

      } catch (HibernateException e1) {
        e1.printStackTrace();
      }

    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
    return book;
  }

  /**
   * sets the book as borrowed to the customer  specified in the database
   * @param primaryKey
   * @param customerPrimaryKey
   */
  public void borrowBook(Integer primaryKey, Integer customerPrimaryKey) {
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Book book = (Book) session.get(Book.class, primaryKey);
      Customer customer = (Customer) session.get(Customer.class,
          customerPrimaryKey);
      if (book != null && customer != null)
        book.setCustomer(customer);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [lalilуna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
```

```
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }

  /**
   * customer returns a book, relation in the db between customer and book is
deleted
   * @param primaryKey
   */
  public void returnBook(Integer primaryKey) {
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;
    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Book book = (Book) session.get(Book.class, primaryKey);

      if (book != null) // session.get returns null when no entry is found
          book.setCustomer(null);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }

    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }

  /**
   * updates/creates a book
   * @param bookValue
   */
  public void saveBook(Book bookValue) {

    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;
    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Book book;
      if (bookValue.getId() != null && bookValue.getId().intValue() != 0) { //
[laliluna] 04.12.2004 load book from DB
        book = (Book) session.get(Book.class, bookValue.getId());
        if (book != null) {
```

```
          book.setAuthor(bookValue.getAuthor());
          book.setTitle(bookValue.getTitle());
          book.setAvailable(bookValue.getAvailable());
          session.update(book);
        }
      }
      else // [laliluna] 04.12.2004 create new book
      {
        book = new Book();
        book.setAuthor(bookValue.getAuthor());
        book.setTitle(bookValue.getTitle());
        book.setAvailable(bookValue.getAvailable());
        session.save(book);
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }

    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }

  /**
   * deletes a book
   * @param primaryKey
   */
  public void removeBookByPrimaryKey(Integer primaryKey) {
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Book book = (Book) session.get(Book.class, primaryKey);
      if (book != null) session.delete(book);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }
```

```
  /**
   * returns all customers from the db
   * @return
   */

  public Customer[] getAllCustomers() {
    /* will hold the books we are going to return later */
    List customers = new ArrayList();
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Query query = session
          .createQuery("select c from Customer as c order by c.name");
      for (Iterator iter = query.iterate(); iter.hasNext();) {
        customers.add((Customer) iter.next());
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
    return (Customer[]) customers.toArray(new Customer[0]);
  }

  /**
   * gets a customer from the db
   * @param primaryKey
   * @return the customer class or  null, when no customer is found
   */
  public Customer getCustomerByPrimaryKey(Integer primaryKey) {
    /* holds our return value */
    Customer customer = null;
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      customer = (Customer) session.get(Customer.class, primaryKey);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
```

```
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
    return customer;
  }

  /**
   * saves the customers to the db
   * @param customer
   */
  public void saveCustomer(Customer customer) {
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;
    try {
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();
      tx = session.beginTransaction();
      if (customer.getId() == null || customer.getId().intValue() == 0) //
[laliluna] 06.12.2004 create customer
        session.save(customer);
      else {
        Customer toBeUpdated = (Customer) session.get(Customer.class, customer
            .getId());
        toBeUpdated.setAge(customer.getAge());
        toBeUpdated.setLastname(customer.getLastname());
        toBeUpdated.setName(customer.getName());
        session.update(toBeUpdated);
      }
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [laliluna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }

    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }

  /**
   * deletes a customer from the database
   * @param primaryKey
   */
  public void removeCustomerByPrimaryKey(Integer primaryKey) {
    /* a Hibernate session */
    Session session = null;
    /* we always need a transaction */
    Transaction tx = null;

    try {
```

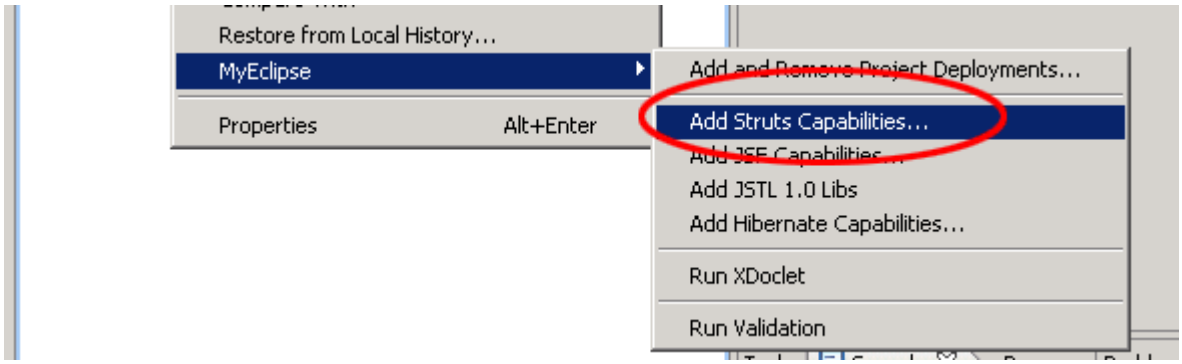
```
      /* get session of the current thread */
      session = HibernateSessionFactory.currentSession();

      tx = session.beginTransaction();
      Customer customer = (Customer) session.get(Customer.class, primaryKey);
      if (customer != null) session.delete(customer);
      tx.commit();
    } catch (HibernateException e) {
      e.printStackTrace();
      // [lalilуna] 17.12.2004 it is recommended to roll back the transaction
after an error occured
      if (tx != null) try {
        tx.rollback();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    } finally {
      try {
        if (session != null) session.close();
      } catch (HibernateException e1) {
        e1.printStackTrace();
      }
    }
  }

}
```

That's all we have created our business logic.

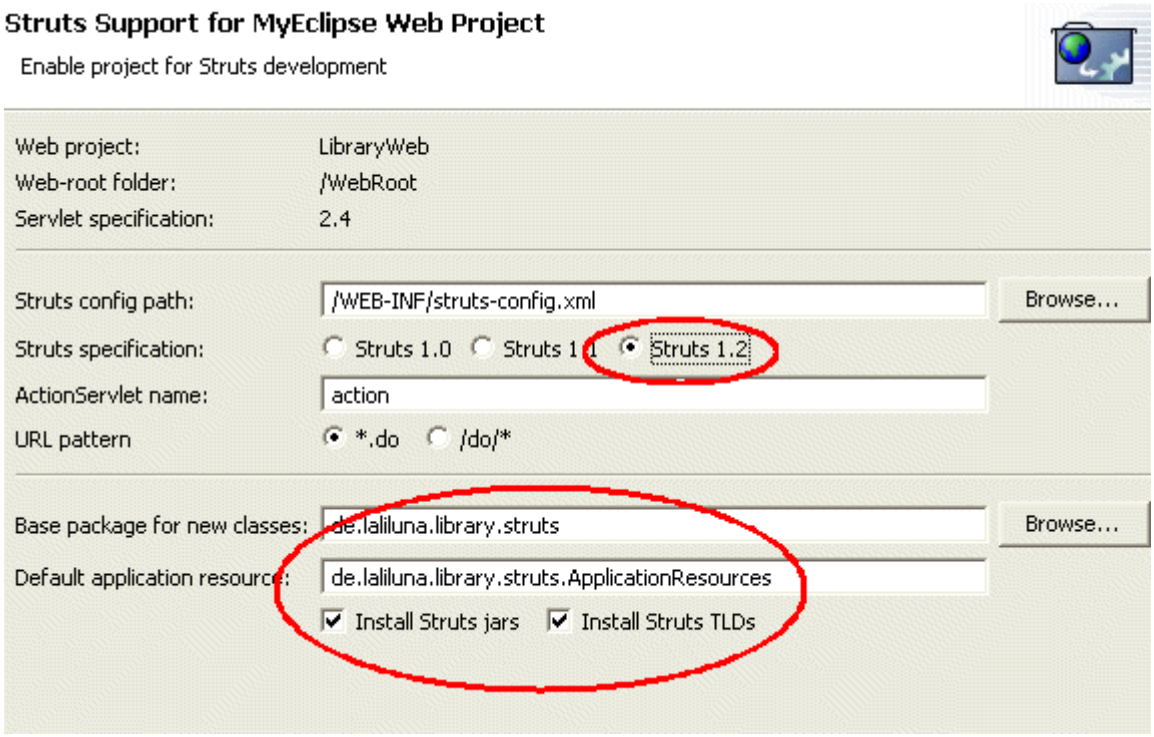
And now the last part: the dialogs

Creating the dialogs with Struts

For now your project is still a normal Web 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 **Base package** for new classes and the **Default application resource**.



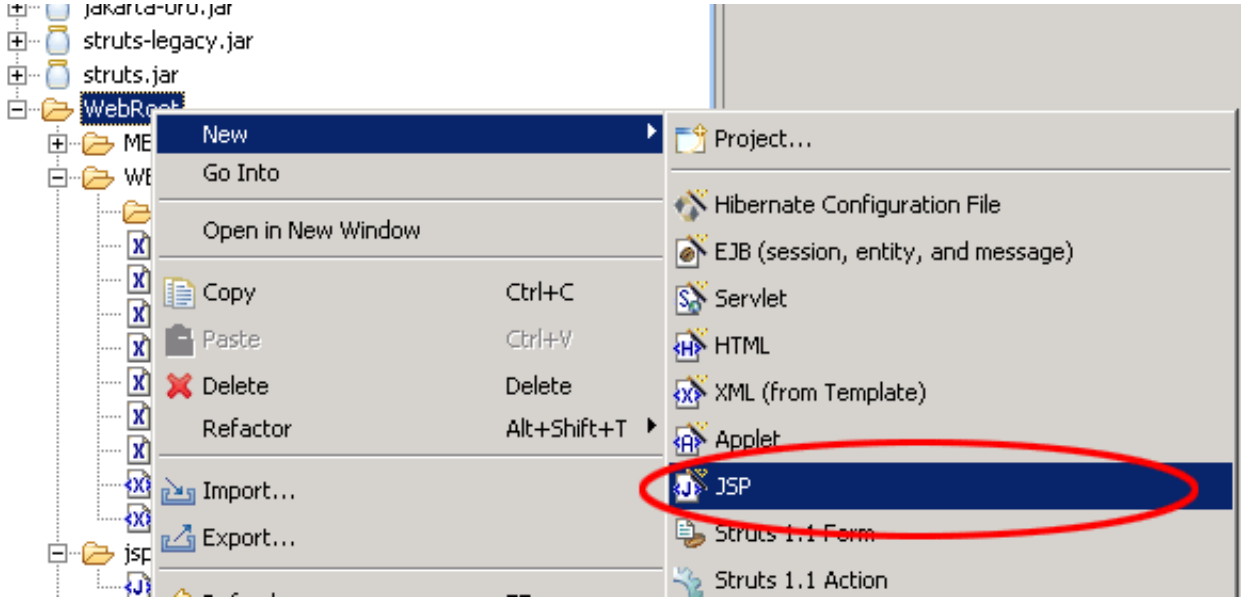
Notice

You can use the libraries and tlds found in the struts-blanc.war when you do not have MyEclipse. Download struts from

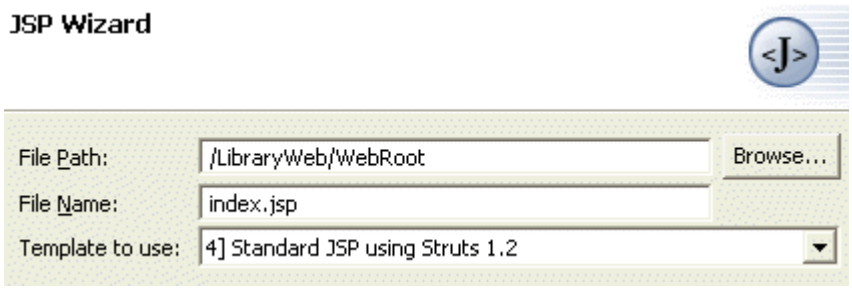
You can find the struts-blank.war in the folder jakarta-struts-1.2.4/webapps.

Create a default, welcome page

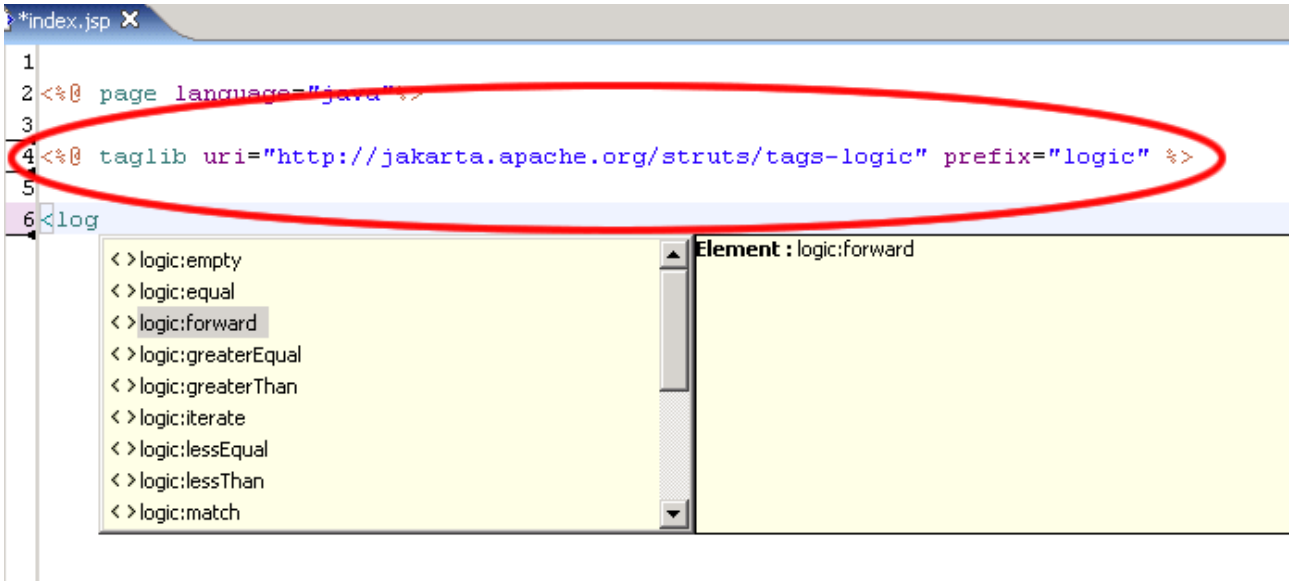
Ok, now we want to create a default page. Right click (yes again) on the Folder `WebRoot` in the Project and choose `New > JSP`.



Set the name to `index.jsp` and choose on `template to use > Standard JSP using Struts 1.2` MyEcplise 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`
Change the body of the file to the following:

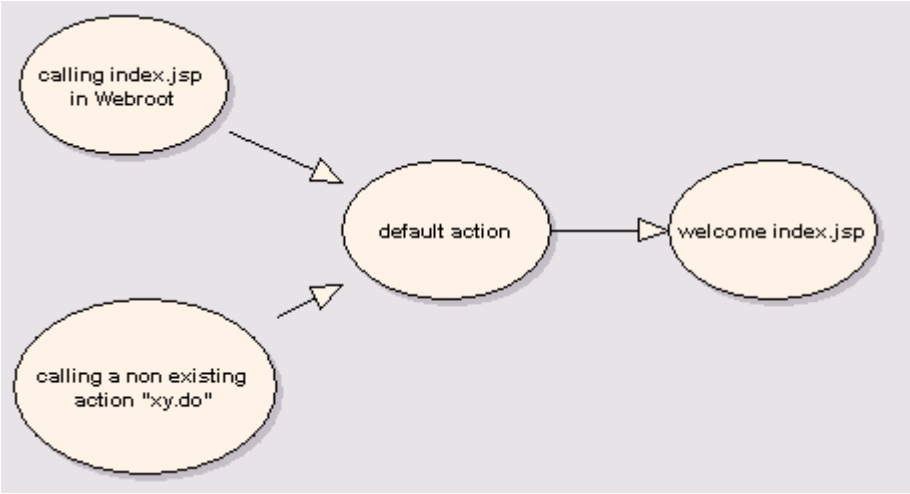
```
  <body>
    Welcome!
    <br>
    <html:link action="bookList">Show the book list</html:link>
    <br>
    <html:link action="customerList">Show the customer list</html:link>
  </body>
```

Global Action Forwards and Action Mappings

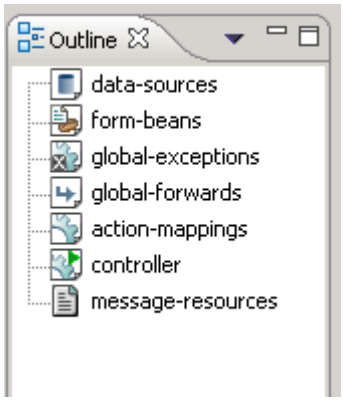
What is an action forward?
A action forward can be used to forward to a jsp or action mapping. There are two different action forwards. The global action forward and the local action forward. You can access a global action forward on each jsp or action class. A local action forward can only be accessed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

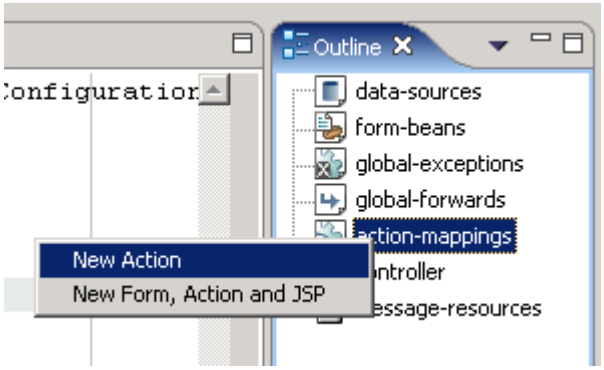


In the first step we create a new action mapping. Open the `struts-config.xml`, you will find it in the folder `WebRoot/WEB-INF`. Right click in the outline view on `action-mapping`.

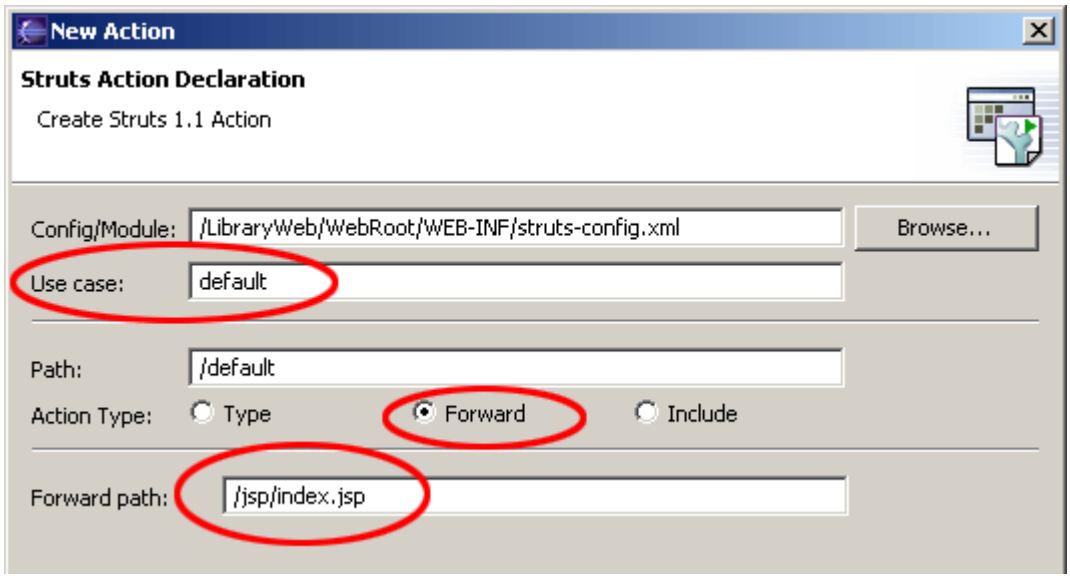


MyEclipse provides some nice features for creating struts files. Open the struts-config.xml and the Outline View.

Click with the right mouse button on the entry action-mappings to create a new action with the wizard.



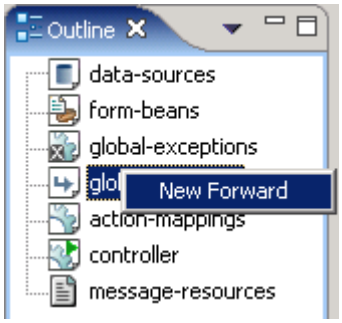
Choose **Use Case** default and **Action Type** Forward. The **Forward Path** is the welcome page / jsp/index.jsp



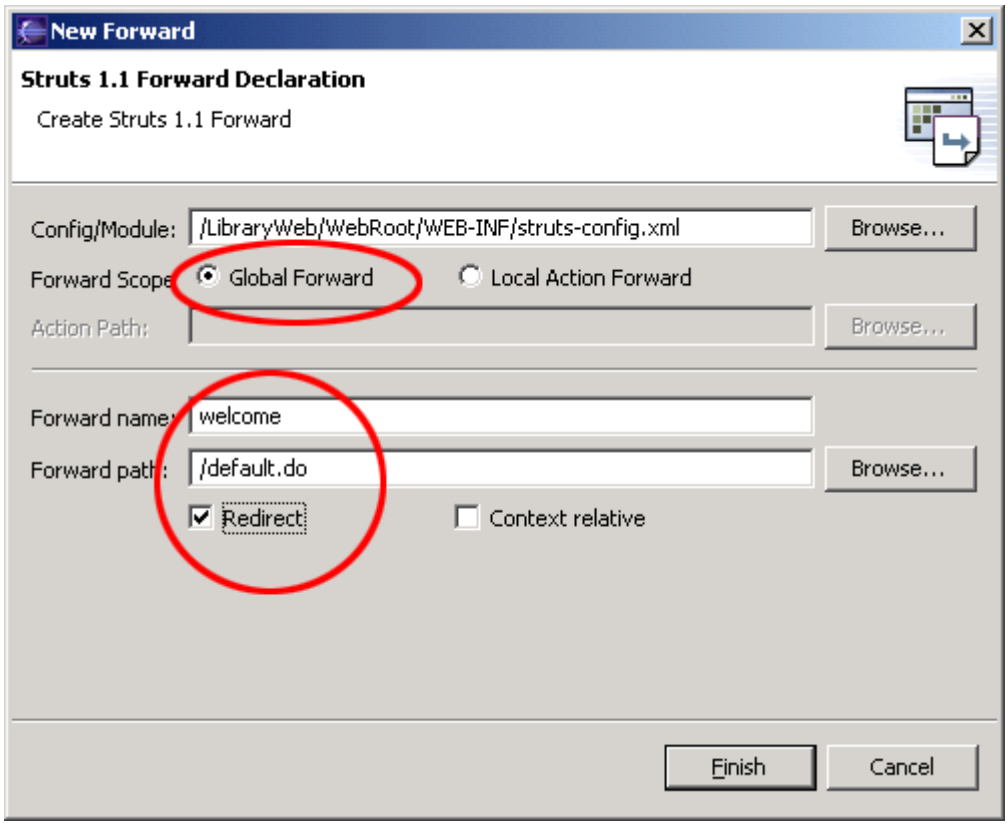
To catch all requests of non existing action mappings, we have to add manually a parameter `unknow="true"` to the action forward.

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

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 struts-config.xml now:

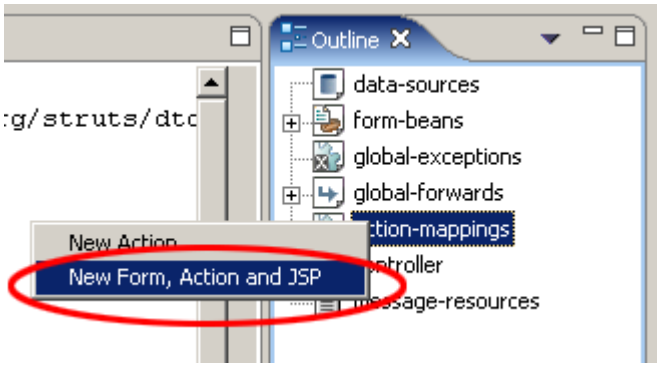
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>
   <data-sources />
   <form-beans />
   <global-exceptions />
   <global-forwards >
      <forward
         name="welcome"
         path="/default.do"
         redirect="true" />

   </global-forwards>

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

```
    </action-mappings>

    <message-resources
parameter="de.laliluna.library.struts.ApplicationResources" />
</struts-config>
```

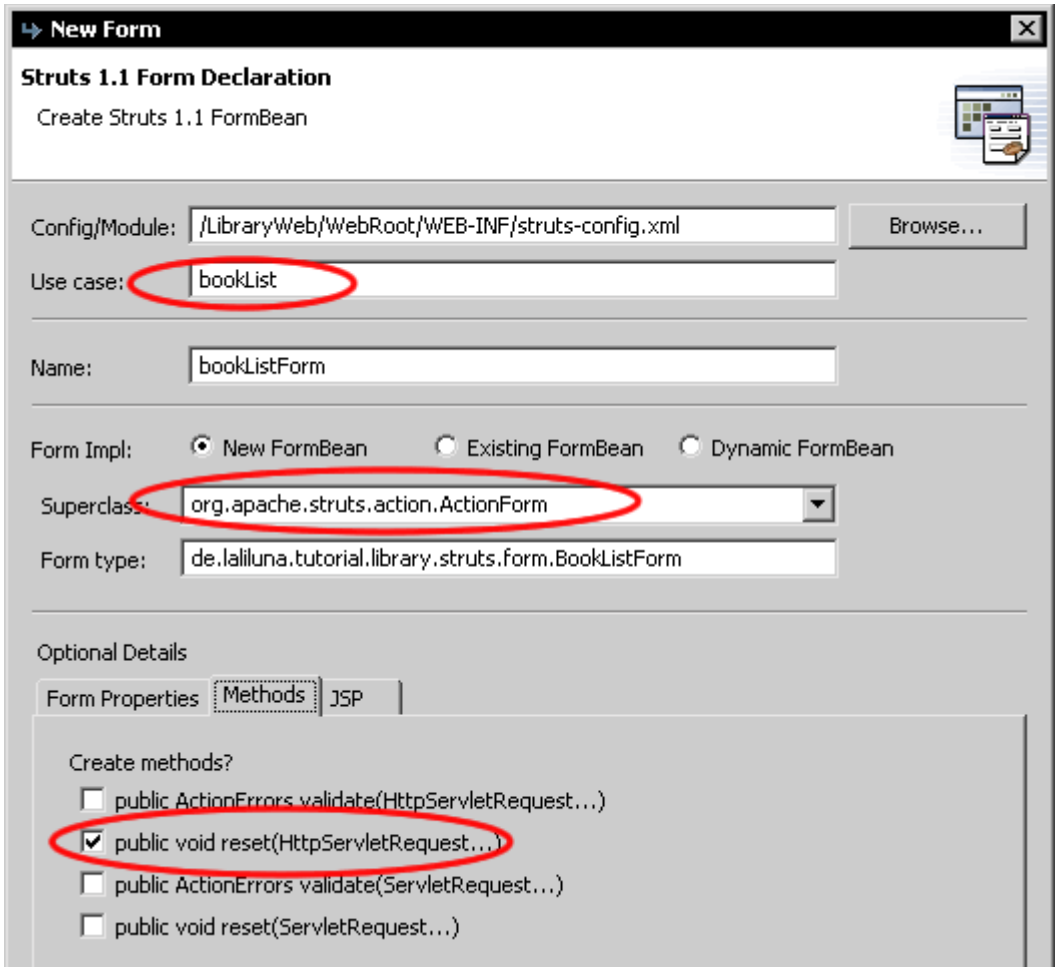
Book list

This use case lists all available books.

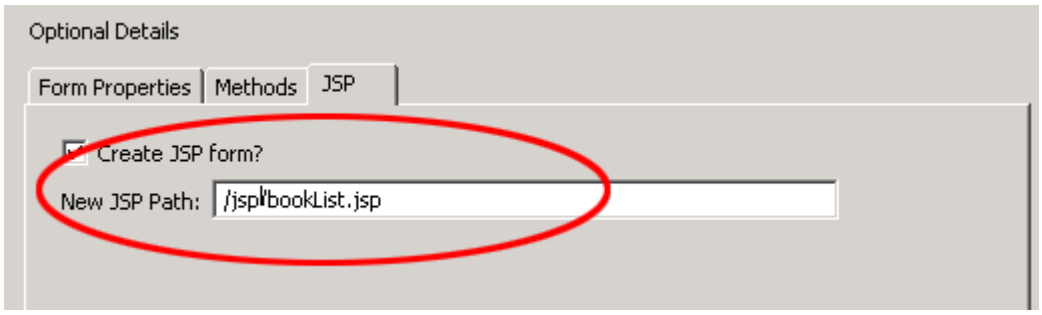
Select the wizard for creating a new form, action and JSP.



Use Case is **bookList**, Superclass **org.apache.struts.ActionForm**. You should create a reset method to initialize your fields. Select **public void reset** to create this method.



Go on to the jsp tab and set the name of the jsp to be created.

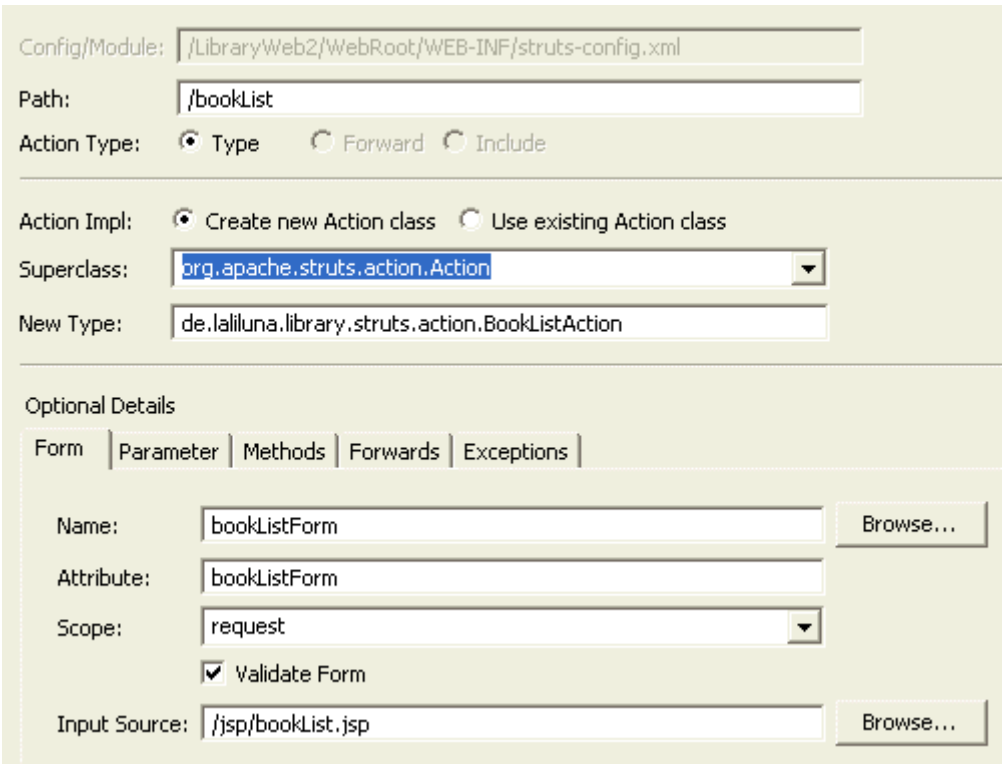


Press the next button to continue to the action mapping.

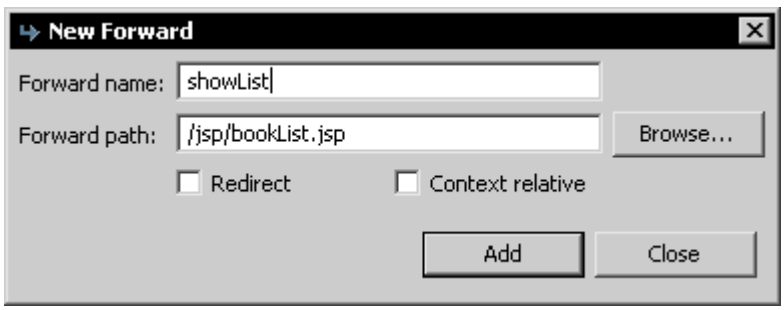
Action mapping und action class of the book list

Make the following changes for the 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.



That's it. Let the files be generated.

Notice

MyEclipse adds the action mapping to the struts-config, creates a blanc JSP, an Action class and the ActionForm class. You can of course do this by hand or using a tool like the struts console.

The action look like:

```
      <action
         attribute="bookListForm"
         input="/jsp/bookList.jsp"
         name="bookListForm"
         path="/bookList"
         scope="request"
         type="de.laliluna.library.struts.action.BookListAction">
         <forward name="showList" path="/jsp/bookList.jsp" />
      </action>
```

Edit the source code of the action form class

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

```
public class BookListForm extends ActionForm
{
  private Book[] book = new Book[0];
  /**
   * @return Returns the book.
   */
  public Book[] getBooks() {
    return book;
  }
  /**
   * @param book The book to set.
   */
  public void setBooks(Book[] bookValues) {
    this.book = bookValues;
  }
  /**
   * Method reset
   * @param mapping
   * @param request
   */
  public void reset(ActionMapping mapping, HttpServletRequest request) {
    book = new Book[0];
  }
}
```

You do not need to type the getter and setter methods. Click with the right mouse button on the project -> select Source -> Generate Getters/Setters.

Edit the source code of the action class

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

Open the class `bookListAction` and edit the method `execute`. 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`.

```
public class BookListAction extends Action
{
  /**
   * Method loads book from DB
   * @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;
    // [lalil una] 27.11.2004 get busines logic
    LibraryManager libraryManager = new LibraryManager();
    // [lalil una] 29.11.2004 update the form bean, from which the jsp will read
the data later.
    bookListForm.setBooks(libraryManager.getAllBooks());
    return mapping.findForward("showList");
  }
}
```

Display the books list in the jsp file.

Open the `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>Borrow by</td>
            <td> </td>
            <td> </td>
            <td> </td>
      </tr>
      <%-- start with an iterate over the collection books --%>
      <logic:iterate name="bookListForm" property="books" id="book">
      <tr>
            <%-- 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>
                  <%-- check if a customer borrowed a book,
                        when its true display his name
                        otherwise display nothing --%>
                  <logic:notEmpty name="book" property="customer">
                        <bean:write name="book" property="customer.name" />,
                        <bean:write name="book" property="customer.lastname" />
                  </logic:notEmpty>
                  <logic:empty name="book" property="customer">
                        -
                  </logic:empty>
```

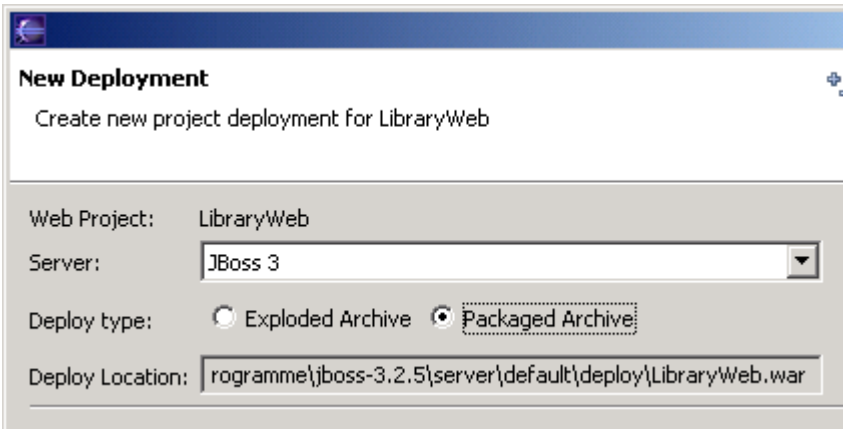
```
            </td>
            <%-- borrow, edit and delete link for each book --%>
            <td>
                  <%-- check if a customer  borrowed a book,
                         when its true display the return link
                         otherwise display the borrow link --%>
                  <logic:notEmpty name="book" property="customer">
                        <html:link action="bookEdit.do?do=returnBook"
                                                paramName="book"
                                                paramProperty="id"
                                                paramId="id">Return
book</html:link>
                  </logic:notEmpty>
                  <logic:empty name="book" property="customer">
                        <html:link action="bookEdit.do?do=borrowBook"
                                                paramName="book"
                                                paramProperty="id"
                                                paramId="id">Borrow
book</html:link>
                  </logic:empty>
            </td>
            <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>
      <%-- end interate --%>

      <%-- if books cannot be found display a text --%>
      <logic:notPresent name="book">
            <tr>
                  <td colspan="5">No books found.</td>
            </tr>
      </logic:notPresent>

      </tbody>
      </table>

      <br>
      <%-- add and back to menu button --%>
      <html:button property="add"
                        onclick="location.href='bookEdit.do?do=addBook'">Add a
new book
      </html:button>
       
      <html:button property="back"
                        onclick="location.href='default.do'">Back to menu
      </html:button>
      </body>
</html>
```

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 customer has borrowed a book or not.

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.

Test the application

Start the jboss and deploy the project LibraryWeb as packaged archiv.



Call the project in your favorite web browser.

http://localhost:8080/LibraryWeb/

Jboss deployment problem

When you redeploy a project Jboss locks very often the libraries. The result is that you get the following message when redeploying.

Undeployment failure on Jboss. File ....jar Unable to be deleted.

An easy solution to this problem is to create two projects, one including the libraries that you don't have to redeploy, the other including your Hibernate Project. We have a tutorial explaining this. If you become anoyed when redeploying, try this tutorial.

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

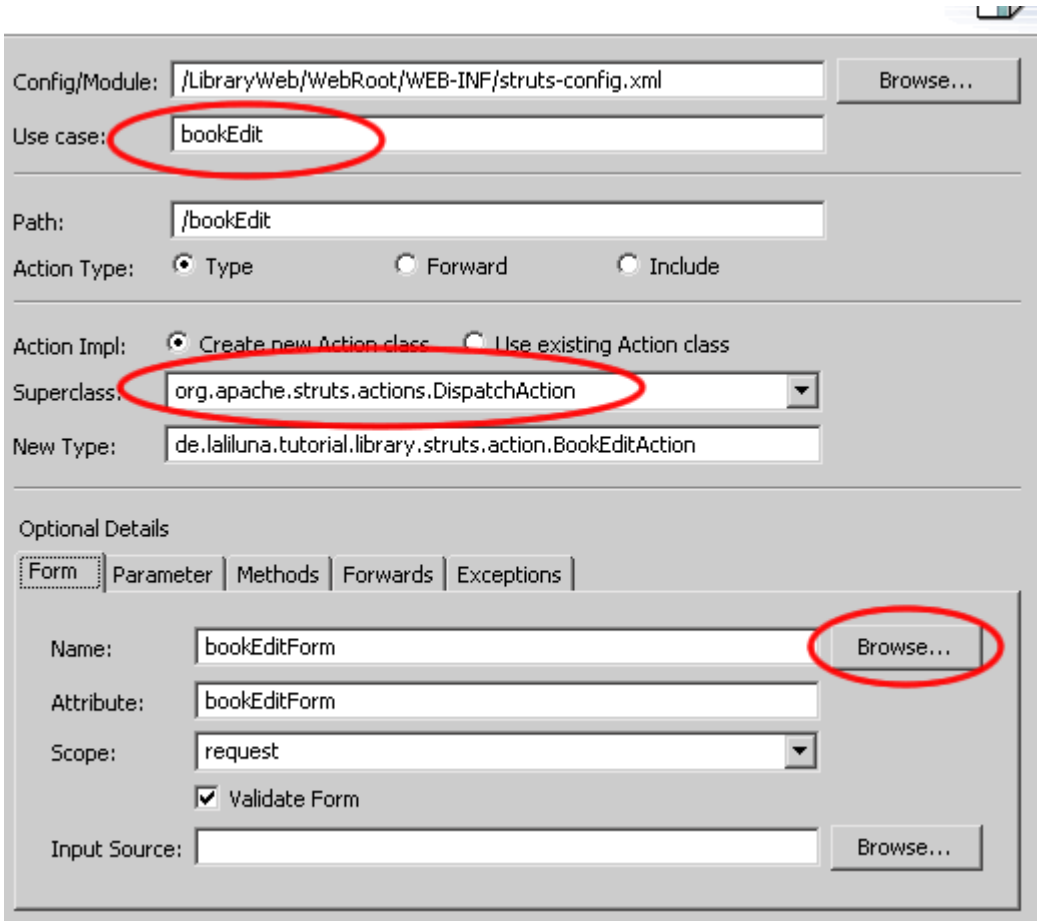
Action Mapping

Create a new action mapping and get the JSP, the actionForm and the action created.

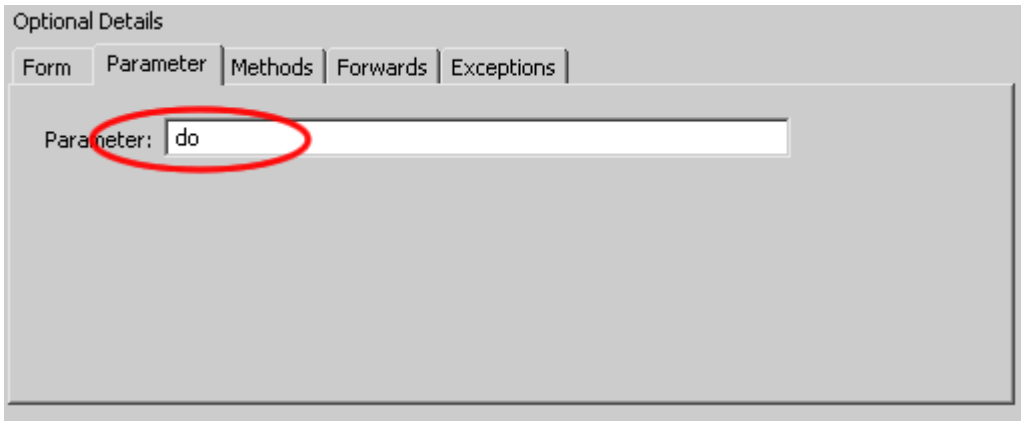
There is a difference to our first action class. The new action class will extend the superclass **org.apache.struts.DispatchAction**. A Dispatch action does not call the execute method but different methods specified by a parameter.

So we can create the following logic:

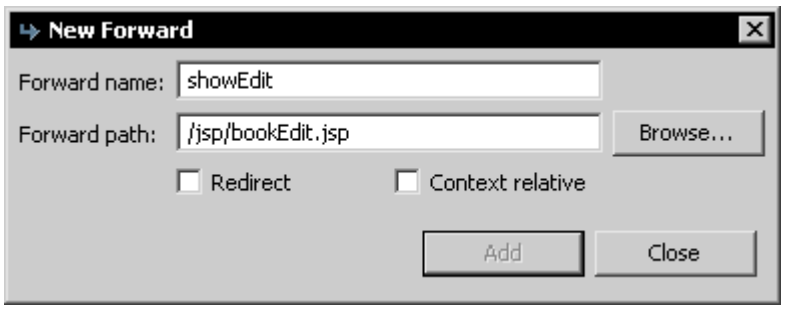
When the customer clicks on an Edit Link the dispatch action will call an Edit method, when he clicks on a create link, the dispatch action calls a create method.

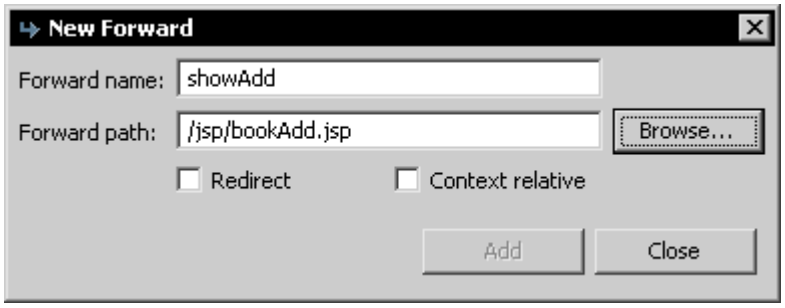


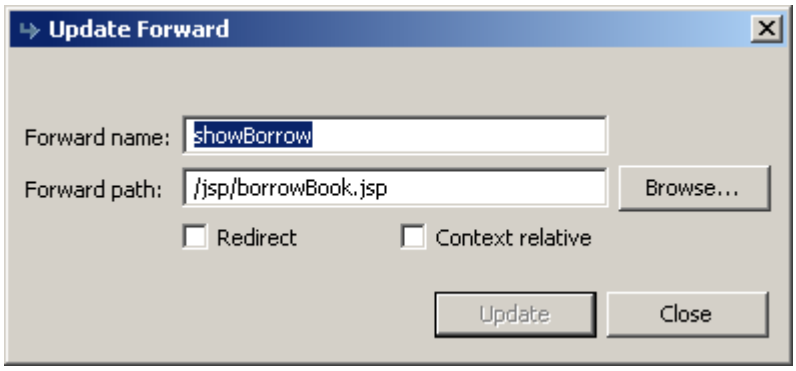
On Parameter we add a parameter `do`. These parameter is needed by the dispatch action class.

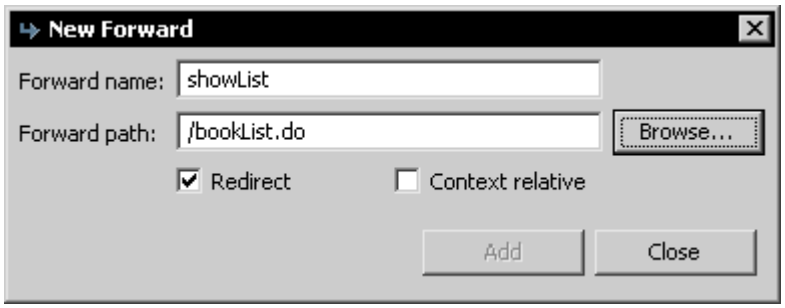


Add four new forwards. One is for the edit page, the second for the add page, where you can add the books, the third forward to the borrow page and the last forward redirect the customer to the book listing.









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

The action mapping looks like:

```
<action
   attribute="bookEditForm"
   input="/jsp/bookEdit.jsp"
   name="bookEditForm"
   parameter="do"
   path="/bookEdit"
   scope="request"
   type="de.laliluna.library.struts.action.BookEditAction">
   <forward name="showBorrow" path="/jsp/borrowBook.jsp" />
   <forward name="showEdit" path="/jsp/bookEdit.jsp" />
   <forward
```

```
        name="showList"
        path="/bookList.do"
        redirect="true" />
    <forward name="showAdd" path="/jsp/bookAdd.jsp" />
  </action>
```

and the form bean looks like:

```
<form-bean name="bookEditForm"
type="de.laliluna.library.struts.form.BookEditForm" />
```

Create the non existing JSP files in the folder jsp with New > JSP.

bookAdd.jsp

bookEdit.jsp

bookBorrow.jsp

Edit the source code of 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">
                                <tbody>
                                <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>
                                </tbody>
                        </table>
                        <%-- set the parameter for the dispatch action --%>
                        <html:hidden property="do" value="saveBook" />

                        <br>
                        <%-- submit and back button --%>
                        <html:button property="back"
                                                onclick="history.back();">
                                                Back
                        </html:button>
                         
                        <html:submit>Save</html:submit>
                </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 fields for id --%>
<html:hidden property="id" />
```

Open the file **bookBorrow.jsp** and add the following.

```
<%@ 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 customers</title>
        </head>
        <body>
        <html:form action="bookEdit">
        <table border="1">
        <tbody>
        <%-- set the header --%>
        <tr>
                <td>Last name</td>
                <td>Name</td>
                <td>Borrow</td>
        </tr>

        <%-- start with an iterate over the collection customer --%>
        <logic:present name="customers">
        <logic:iterate name="customers" id="customer">
        <tr>
                <%-- book informations --%>
                <td><bean:write name="customer" property="lastname" /></td>
                <td><bean:write name="customer" property="name" /></td>
                <td><html:radio property="customerId" value="${customer.id}" /
></td>
        </tr>
        </logic:iterate>
        </logic:present>
        <%-- end interate --%>

        <%-- if customers cannot be found display a text --%>
        <logic:notPresent name="customers">
                <tr>
                        <td colspan="5">No customers found.</td>
                </tr>
        </logic:notPresent>
        </tbody>
        </table>

        <%-- set the book id to lent --%>
        <html:hidden property="id" />

        <%-- set the parameter for the dispatch action --%>
        <html:hidden property="do" value="saveBorrow" />

        <%-- submit and back button --%>
```

```
        <html:button property="back"
                                    onclick="history.back();">
                                    Back
        </html:button>
         
        <html:submit>Save</html:submit>
        </html:form>
        </body>
</html>
```

Form bean

Open the class **BookEditForm.java** in **de.laliluna.library.struts.form** .

Notice

To create an actionForm you normally add all the fields you need to the action form and copy than later to a business object to save them. In our case this would be copying the fields from the BookEditForm to a Book object.

The approach used here adds a object from our business logic (Book) as field to the actionForm. Then we create delegate methods for each field, for example:

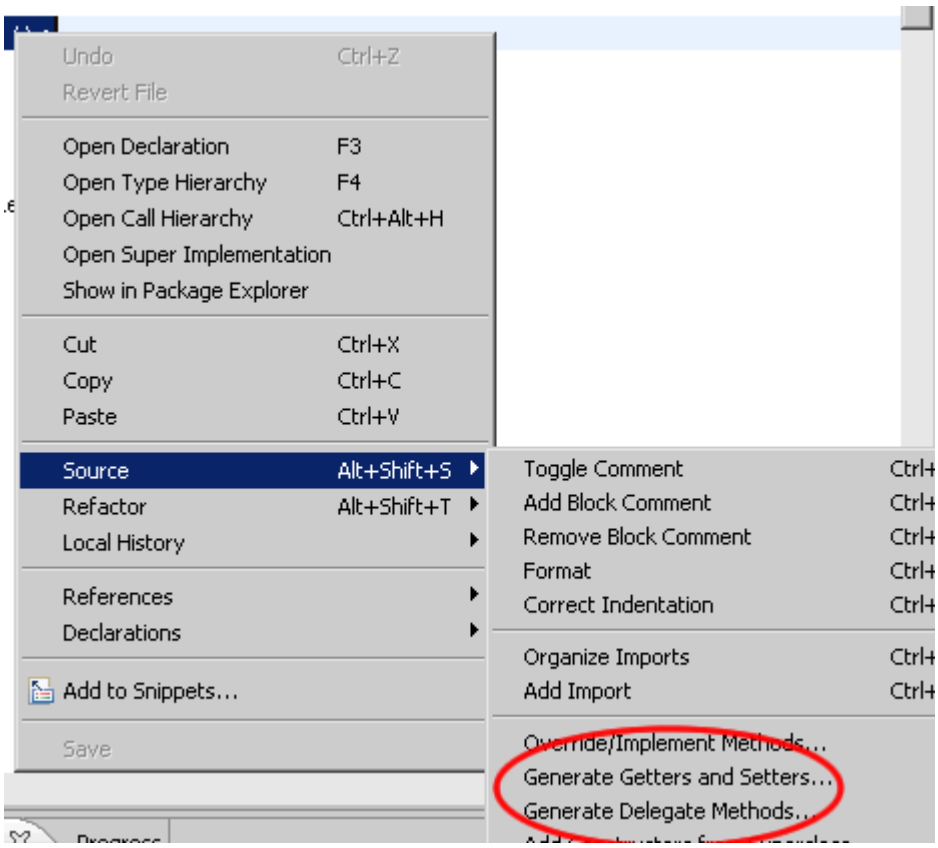
```
public Boolean getAvailable() {
  return book.getAvailable();
}
```

This approach is considered by some people to be not a good design, as we bypass the idea of the Struts design pattern to separate business logic and the dialogs. The advantage is that is quite fast to develop.

Create the attributes book and customerId .

```
public class BookEditForm extends ActionForm {

      private Book book = new Book();

        /**
         * we will need this field to save the customer id in the dialogs where
a customer borrows a book
         */
        private Integer customerId;
```

Generate getters and setters for the attributes.



Then generate all delegate methods for the book attribute.

Finally implement the reset method.

```
public void reset(ActionMapping arg0, HttpServletRequest arg1) {
      book =new Book();
}
```

The source code looks like the following.

```
public class BookEditForm extends ActionForm {
 private Book book;

 /**
 * we will need this field to save the customer id in the dialogs where a
customer borrows a book
 */
 private Integer customerId;

 /**
 * @return Returns the book.
 */
 public Book getBook() {
 return book;
 }
 /**
 * @param book The book to set.
 */
 public void setBook(Book book) {
 this.book = book;
 }
 /* (non-Javadoc)
 * @see java.lang.Object#equals(java.lang.Object)
 */
 public boolean equals(Object arg0) {
 return book.equals(arg0);
 }
      public void reset(ActionMapping arg0, HttpServletRequest arg1) {
 book =new Book();
```

```
}

/**
* @return
*/
 public String getAuthor() {
   return book.getAuthor();
 }
 /**
  * @return
  */
 public Boolean getAvailable() {
   return book.getAvailable();
 }
 /**
  * @return
  */
 public Customer getCustomer() {
   return book.getCustomer();
 }
 /**
  * @return
  */
 public Integer getId() {
   return book.getId();
 }
 /**
  * @return
  */
 public String getTitle() {
   return book.getTitle();
 }
 /* (non-Javadoc)
  * @see java.lang.Object#hashCode()
  */
 public int hashCode() {
   return book.hashCode();
 }
 /**
  * @param author
  */
 public void setAuthor(String author) {
   book.setAuthor(author);
 }
 /**
  * @param available
  */
 public void setAvailable(Boolean available) {
   book.setAvailable(available);
 }
 /**
  * @param customer
  */
 public void setCustomer(Customer customer) {
   book.setCustomer(customer);
 }
 /**
  * @param id
  */
 public void setId(Integer id) {
   book.setId(id);
 }
 /**
  * @param title
  */
 public void setTitle(String title) {
   book.setTitle(title);
 }
 /* (non-Javadoc)
```

```
   * @see java.lang.Object#toString()
   */
  public String toString() {
    return book.toString();
  }
  /**
   * @return Returns the customerId.
   */
  public Integer getCustomerId() {
    return customerId;
  }
  /**
   * @param customerId The customerId to set.
   */
  public void setCustomerId(Integer customerId) {
    this.customerId = customerId;
  }
}
```

Methods of the dispatch action class

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

```
public class BookEditAction extends DispatchAction {
        /**
         * loads the book specified by the id from the database and forwards to
the edit form
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward editBook(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
          System.out.println("editBook");
                BookEditForm bookEditForm = (BookEditForm) form;

                /* lalinuna.de 04.11.2004
                 * get id of the book from request
                 */
                Integer id = Integer.valueOf(request.getParameter("id"));
                        // [lalilуna] 28.11.2004  get business logic
                LibraryManager libraryManager = new LibraryManager();
                bookEditForm.setBook(libraryManager.getBookByPrimaryKey(id));
                return mapping.findForward("showEdit");
        }

        /**
         * loads a book from the db and forwards to the borrow book form
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward borrowBook(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
          System.out.println("borrowBook");

          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 customer
                 * get the book information and get all customers
                 */
                LibraryManager libraryManager = new LibraryManager();

                // [laliluna] 28.11.2004 save book in the form
                bookEditForm.setBook(libraryManager.getBookByPrimaryKey(id));
                // [lalilуna] 28.11.2004 save customers in the reqest
                request.setAttribute("customers", libraryManager.getAllCustomers
());

                return mapping.findForward("showBorrow");
        }

        /**
         * return a book from a customer
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward returnBook(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
          System.out.println("returnBook");

          BookEditForm bookEditForm = (BookEditForm) form;

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

                // [lalilуna] 28.11.2004 get business logic
                LibraryManager libraryManager = new LibraryManager();

                libraryManager.returnBook(id);

                return mapping.findForward("showList");
        }

        /**
         * deletes a book from the database
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward deleteBook(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
          System.out.println("deleteBook");

          BookEditForm bookEditForm = (BookEditForm) form;

                /* lalinuna.de 04.11.2004
                 * get id of the book from request
```

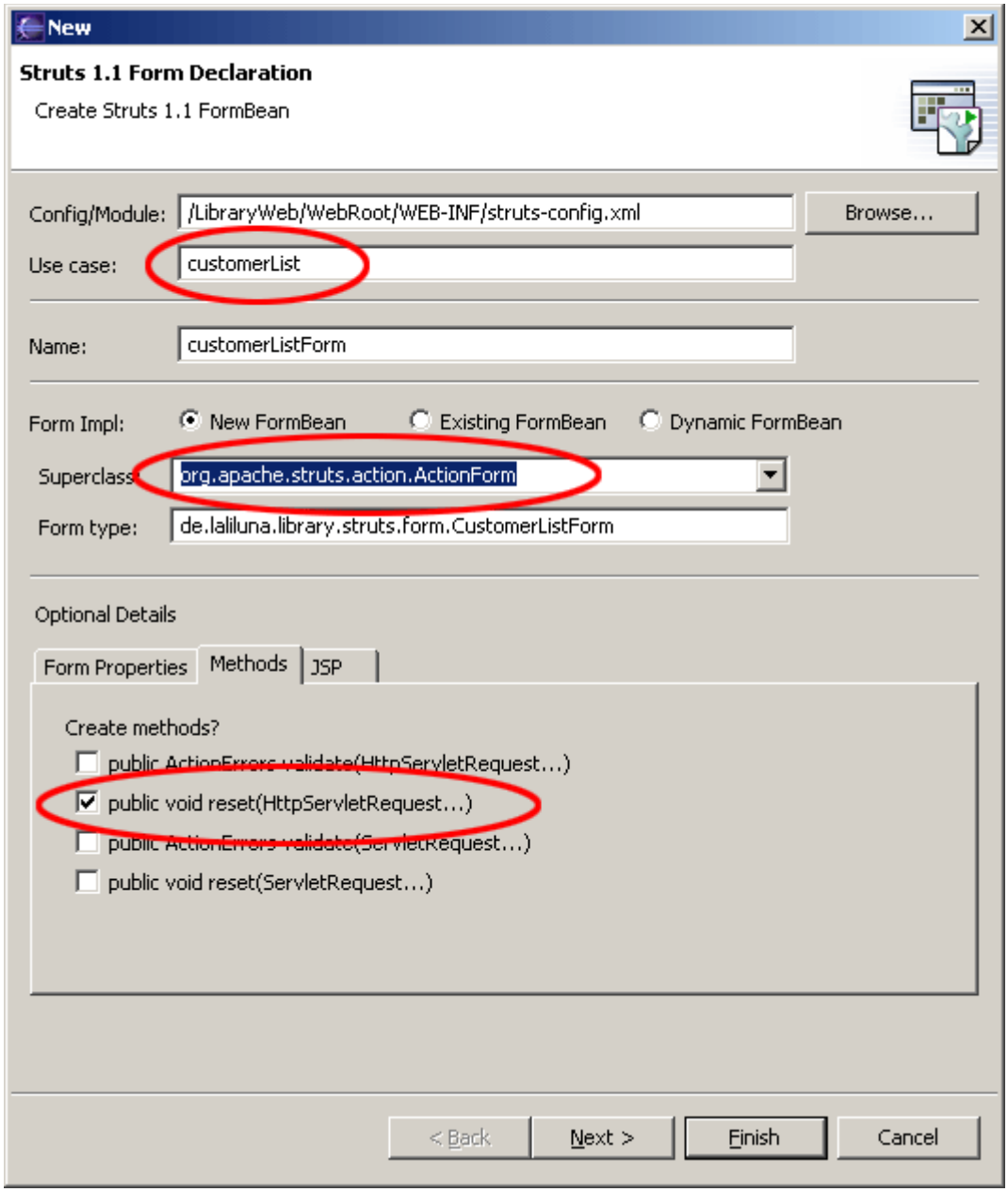
```
                 */
                Integer id = Integer.valueOf(request.getParameter("id"));

                // [laliluna] 28.11.2004 get business logic
                LibraryManager libraryManager = new LibraryManager();

                libraryManager.removeBookByPrimaryKey(id);

                return mapping.findForward("showList");
        }
        /**
         * forwards to the add book form
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward addBook(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
          System.out.println("addBook");

          BookEditForm bookEditForm = (BookEditForm) form;

                return mapping.findForward("showAdd");

        }

        /**
         * saves the borrow assigned in the form in the database
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return ActionForward
         */
        public ActionForward saveBorrow(
                ActionMapping mapping,
                ActionForm form,
                HttpServletRequest request,
                HttpServletResponse response) {
                BookEditForm bookEditForm = (BookEditForm) form;

                // [laliluna] 28.11.2004 get business logc
                LibraryManager libraryManager = new LibraryManager();
                libraryManager.borrowBook(bookEditForm.getId(),
bookEditForm.getCustomerId());

                return mapping.findForward("showList");
        }

        /**
         * updates or creates the book in the database
         * @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;
```

```
                // [lalilуna] 28.11.2004 get business logic
                LibraryManager libraryManager = new LibraryManager();
                libraryManager.saveBook(bookEditForm.getBook());
                return mapping.findForward("showList");
        }
}
```

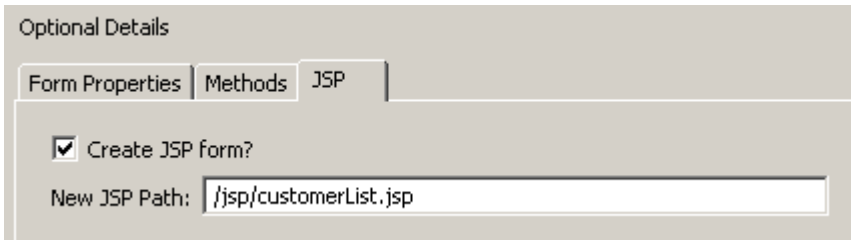
You can redeploy your application now to test all the functions considering editing books.

Use case Customer list

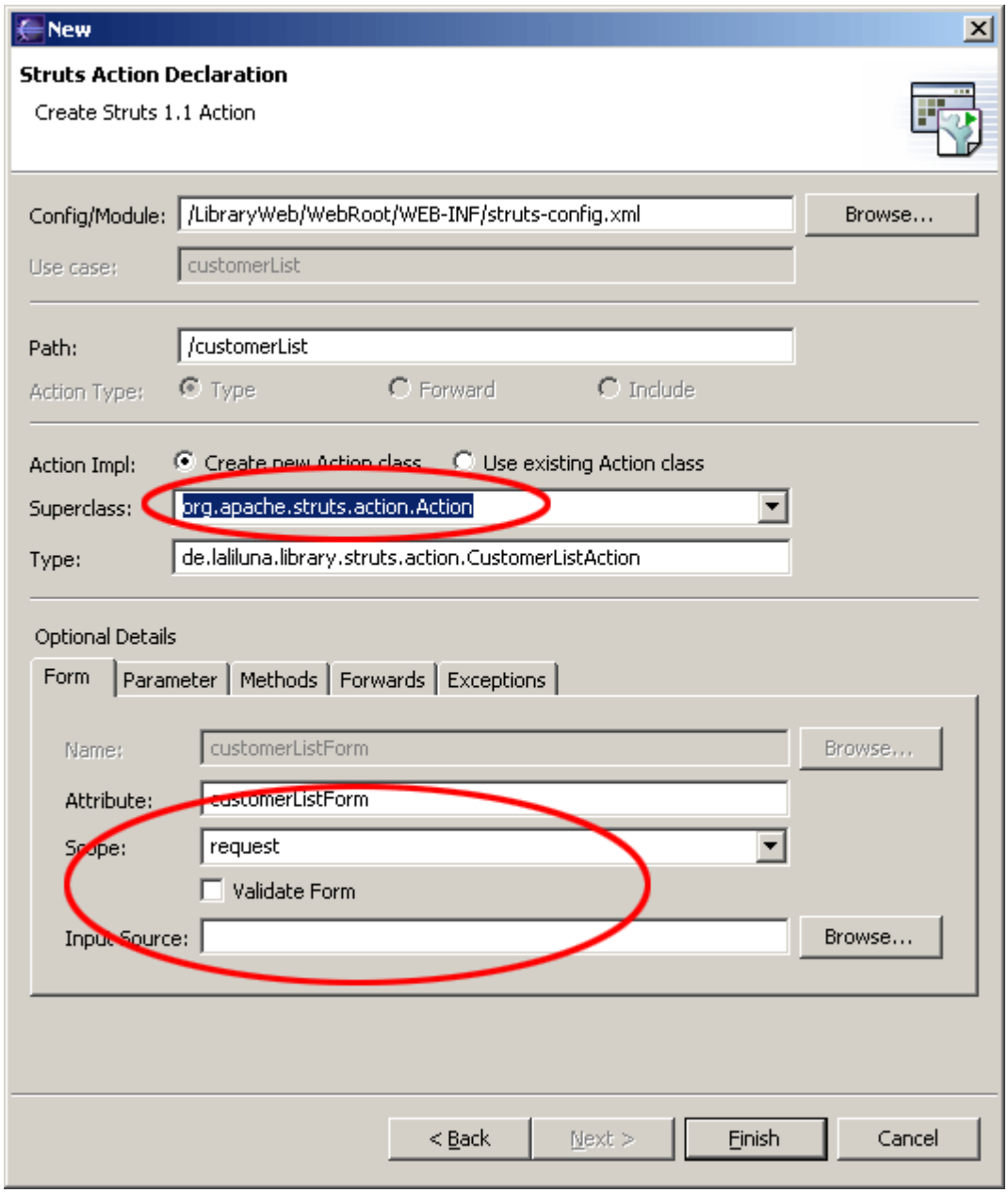
We create this list on the same way like the book list. Open the `struts-config.xml`. Select the wizard to create an action, a form and forwards at the same time. Our use case is customer list. Edit the dialog as shown below:



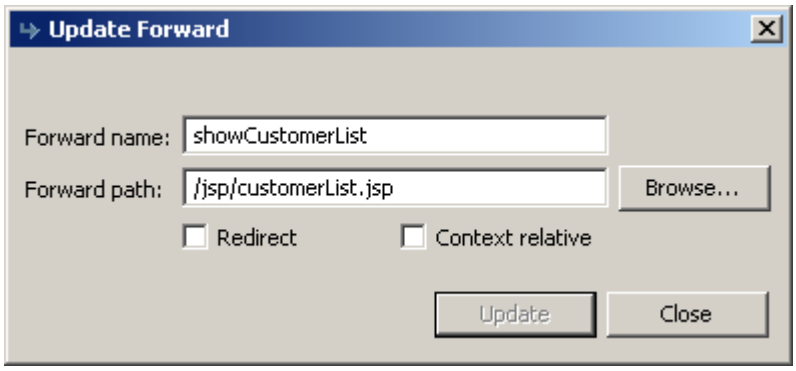
Do not forget the changes on the method tabs. On the JSP tab create the following JSP.



The next step is to set the action which is called before your JSP is shown. Make the changes we marked below.



The last step is to create the forward from the action which will forward to the JSP showing the customer list.



For now, we have created all the files we need for our usecase. The next step is to fill them with content.

For non MyEclipse user here are the action mapping and the form declaration in the struts-config:

```
 <action
      attribute="customerListForm"
      input="/jsp/customerList.jsp"
      name="customerListForm"
      path="/customerList"
      scope="request"
      type="de.laliluna.library.struts.action.CustomerListAction"
      validate="false">
   <forward name="showCustomerList" path="/jsp/customerList.jsp" />
</action>
```

```
<form-bean name="customerListForm"
type="de.laliluna.library.struts.form.CustomerListForm" />
```

Edit the source code of the action form class

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

```
public class CustomerListForm extends ActionForm {
       private Customer[] customers;
       /**
       * @return Returns the customers.
       */
       public Customer[] getCustomers() {
        return customers;
       }
       /**
       * @param customers The customers to set.
       */
       public void setCustomers(Customer[] customers) {
        this.customers = customers;
       }
      /**
       * Method reset
       * @param mapping
       * @param request
       */
      public void reset(ActionMapping mapping, HttpServletRequest request) {
            customers = new Customer[0];
      }
}
```

Edit the action class.

```
public class CustomerListAction extends Action
{
  /**
   * loads customers from the db and saves them in the request
   * @param mapping
   * @param form
   * @param request
   * @param response
   * @return ActionForward
   */
  public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
  {
    CustomerListForm customerListForm = (CustomerListForm) form;
    // [lalil una] 29.11.2004 get business logic
    LibraryManager libraryManager = new LibraryManager();

    customerListForm.setCustomers(libraryManager.getAllCustomers());
    return mapping.findForward("showCustomerList");
  }
}
```

Displaying the custom list

Open the JSP file `customerList.jsp` and change the content of the file to the following.

```
<%@ 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>JSP for customerListForm form</title>
        </head>
        <body>
                        <table border="1">
        <tbody>
        <%-- set the header --%>
        <logic:present name="customerListForm" property="customers">

         <tr>
                <td>Name</td>
                <td>Last name</td>
                <td>Age</td>
                <td></td>
                <td></td>
         </tr>
         <%-- start with an iterate over the collection books --%>
         <logic:iterate name="customerListForm" property="customers"
id="customer">
         <tr>
                 <%-- book informations --%>
                 <td><bean:write name="customer" property="name" /></td>
                 <td><bean:write name="customer" property="lastname" /></td>
                 <td><bean:write name="customer" property="age" /></td>
                <%-- edit and delete link for each customer --%>
                <td><html:link action="customerEdit.do?do=editCustomer"
                                        paramName="customer"
                                        paramProperty="id"
                                        paramId="id">Edit</html:link>
                </td>
                <td><html:link action="customerEdit.do?do=deleteCustomer"
                                        paramName="customer"
```

```
                                        paramProperty="id"
                                        paramId="id">Delete</html:link>
                </td>
         </tr>
         </logic:iterate>
         <%-- end interate --%>
  </logic:present>
        <%-- if customers cannot be found display a text --%>
        <logic:notPresent name="customerListForm" property="customers">
                <tr>
                        <td colspan="5">No customers found.</td>
                </tr>
        </logic:notPresent>

        </tbody>
        </table>

        <br>
        <%-- add and back to menu button --%>
        <html:button property="add"
                                onclick="location.href='customerEdit.do?do=addC
ustomer'">Add a new customer
        </html:button>
         
        <html:button property="back"
                                onclick="location.href='default.do'">Back to
menu
        </html:button>
        </body>
</html>
```

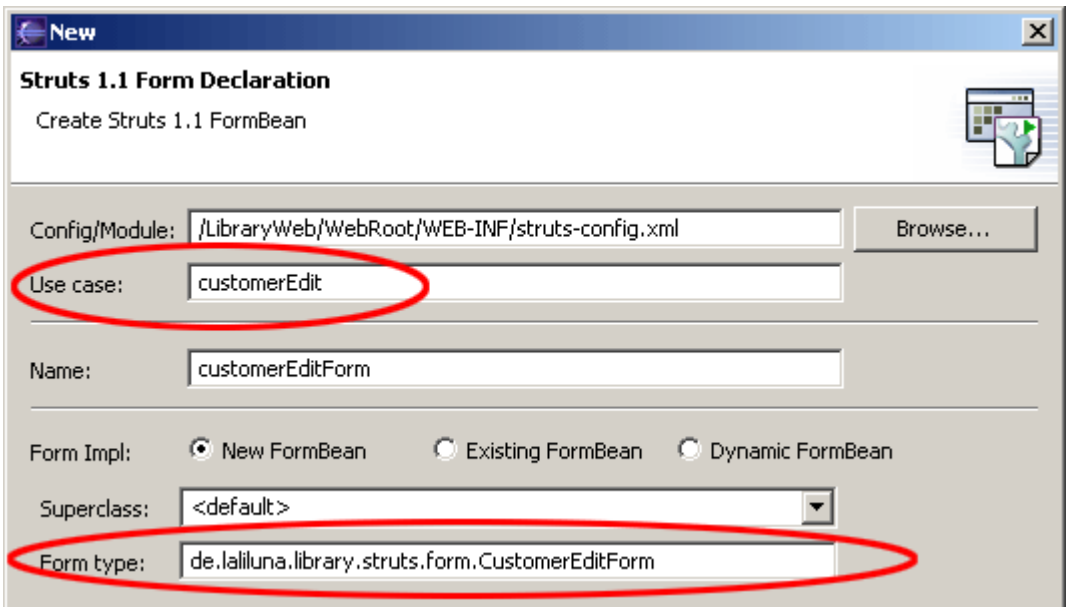
That'it. We have finished the use case. You may test it now.

Use case add, edit, delete customers

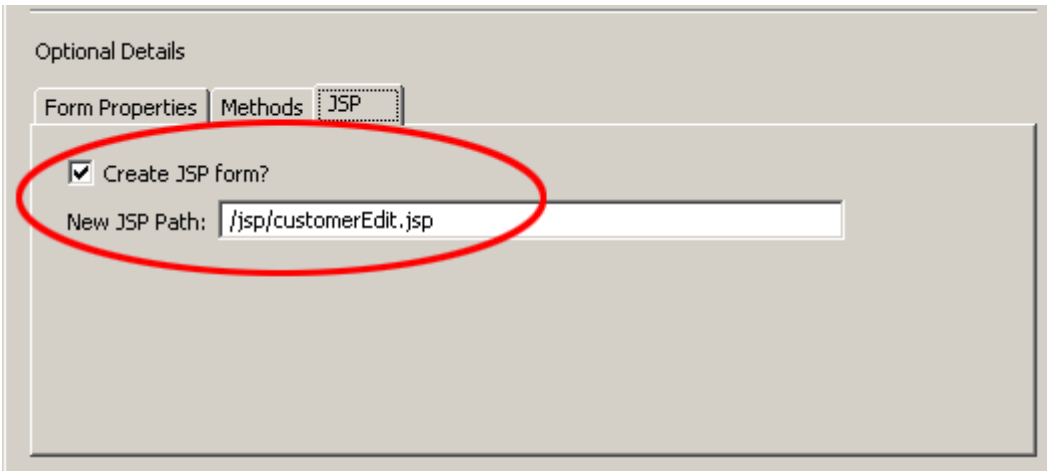
In the next step we want to add the following processes.

- Add a customer
- Edit a customer
- Delete customer

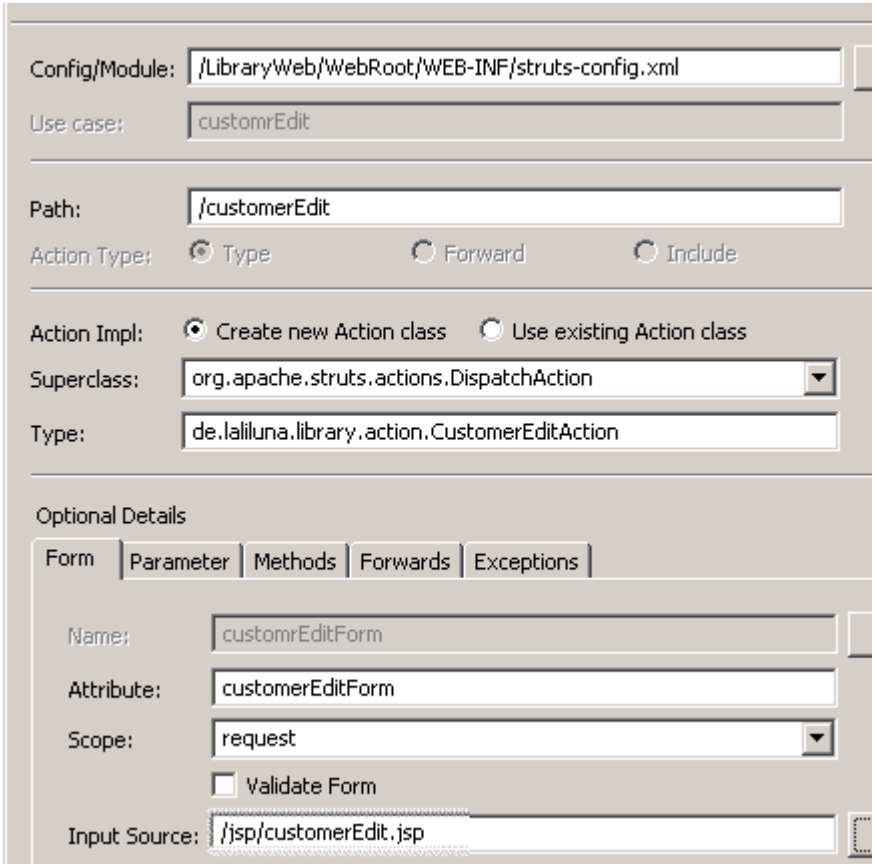
Select „New Form, Action and JSP“.



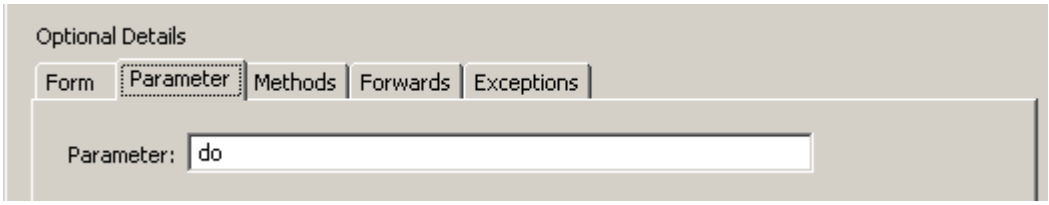
Select to create a JSP file.



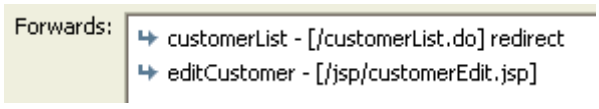
Continue to the action page. Select DispatchAction as the Super Class.



Then select to create a parameter:



Then create two forwards as shown below.



Notice:

For non MyEclipse user: Create the following entries in your struts-config.xml:

```
      <form-bean name="customerEditForm"
type="de.laliluna.library.struts.form.CustomerEditForm" />
```

```
      <action
         attribute="customerEditForm"
         input="/jsp/customerEdit.jsp"
         name="customerEditForm"
         parameter="do"
         path="/customerEdit"
         scope="request"
         type="de.laliluna.library.struts.action.CustomerEditAction"
         validate="false">
         <forward
```

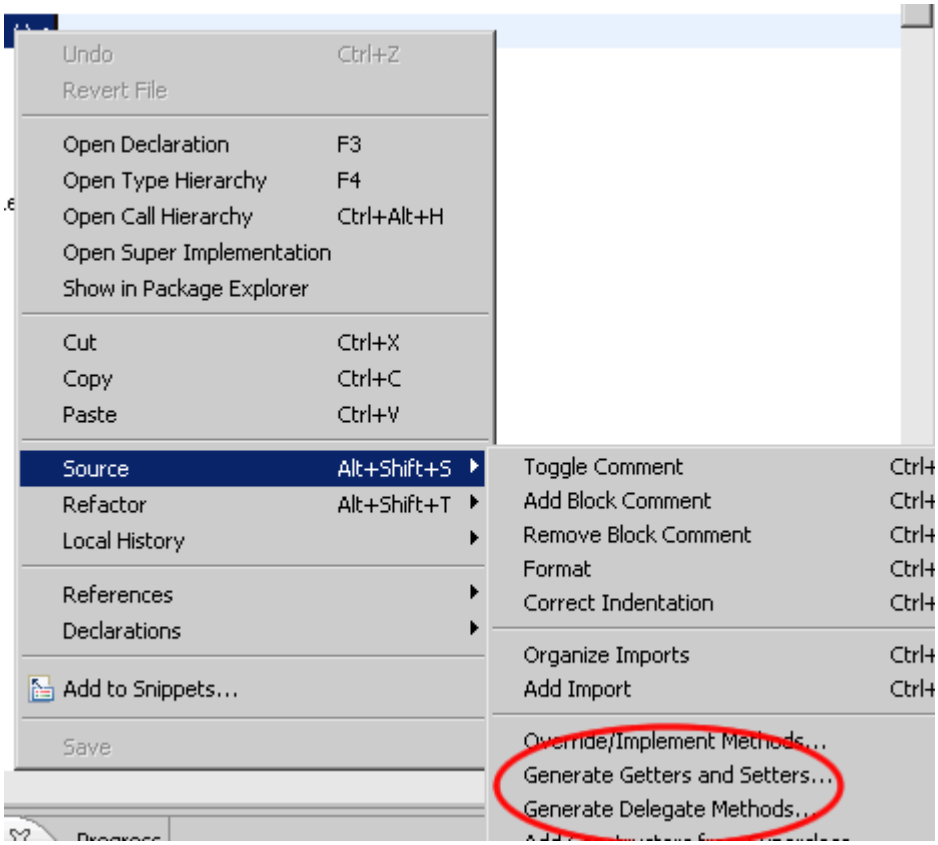
```
            name="customerList"
            path="/customerList.do"
            redirect="true" />
        <forward name="editCustomer" path="/jsp/customerEdit.jsp" />
```

Customer form bean

Add a new attribute of type `Customer`

```
  private Customer customer;
```

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



Finally implement the reset Method.

The source code of the class looks like the following

```
public class CustomerEditForm extends ActionForm {

  private Customer customer;

  /**
   * @return Returns the customer.
   */
  public Customer getCustomer() {
    return customer;
  }
  /**
   * @param customer The customer to set.
   */
  public void setCustomer(Customer customer) {
    this.customer = customer;
  }
  /**
```

```
 * Method reset
 * @param mapping
 * @param request
 */
public void reset(ActionMapping mapping, HttpServletRequest request) {

 customer=new Customer();

}
/* (non-Javadoc)
 * @see java.lang.Object#equals(java.lang.Object)
 */
public boolean equals(Object arg0) {
  return customer.equals(arg0);
}
/**
 * @return
 */
public Integer getAge() {
  return customer.getAge();
}
/**
 * @return
 */
public Integer getId() {
  return customer.getId();
}
/**
 * @return
 */
public String getLastname() {
  return customer.getLastname();
}
/**
 * @return
 */
public String getName() {
  return customer.getName();
}
/* (non-Javadoc)
 * @see java.lang.Object#hashCode()
 */
public int hashCode() {
  return customer.hashCode();
}
/**
 * @param age
 */
public void setAge(Integer age) {
  customer.setAge(age);
}
/**
 * @param id
 */
public void setId(Integer id) {
  customer.setId(id);
}
/**
 * @param lastname
 */
public void setLastname(String lastname) {
  customer.setLastname(lastname);
}
/**
 * @param name
 */
public void setName(String name) {
  customer.setName(name);
}
```

```
  /* (non-Javadoc)
   * @see java.lang.Object#toString()
   */
  public String toString() {
    return customer.toString();
  }
}
```

Edit the source code of the action class

Open the file **CustomerEditAction.class** in the package **de.laliluna.library.struts.action** and add the following methods.

The first one is the step right before editing a customer. It loads the customer data from the database and saves it to the form bean.

```
/**
 * loads customer from the db and forwards to the edit form
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 */
 public ActionForward prepareEdit(ActionMapping mapping, ActionForm form,
 HttpServletRequest request, HttpServletResponse response) {
 CustomerEditForm customerEditForm = (CustomerEditForm) form;

 Integer id = Integer.valueOf(request.getParameter("id"));
 LibraryManager libraryManager = new LibraryManager();

 customerEditForm.setCustomer(libraryManager.getCustomerByPrimaryKey(id));

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

The next method is the step right before the create customer JSP is opened. Actually it does only forward to the JSP.

```
  /**
 * prepares the add form (actually only forwards to it)
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 */
 public ActionForward prepareAdd(ActionMapping mapping, ActionForm form,
 HttpServletRequest request, HttpServletResponse response) {
 return mapping.findForward("editCustomer");

 }
```

The update and creation of customers is made in the next method.

```
  /**
 * saves the customers and forwards to the list
 * @param mapping
 * @param form
 * @param request
 * @param response
 * @return
 */
 public ActionForward saveCustomer(ActionMapping mapping, ActionForm form,
 HttpServletRequest request, HttpServletResponse response) {
 CustomerEditForm customerEditForm = (CustomerEditForm) form;
```

```
 LibraryManager libraryManager = new LibraryManager();
 libraryManager.saveCustomer(customerEditForm.getCustomer());

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

And finally when you click on delete, the following method is called.

```
  /**
   * deletes the customers and forwards to the list
   * @param mapping
   * @param form
   * @param request
   * @param response
   * @return
   */
  public ActionForward deleteCustomer(ActionMapping mapping, ActionForm form,
      HttpServletRequest request, HttpServletResponse response) {
    CustomerEditForm customerEditForm = (CustomerEditForm) form;
    LibraryManager libraryManager = new LibraryManager();
    libraryManager.removeCustomerByPrimaryKey(customerEditForm.getCustomer().
getId());

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

When the business logic is kept separated, the code in the action is always very short and easy to read.

Edit the source code of the jsp file

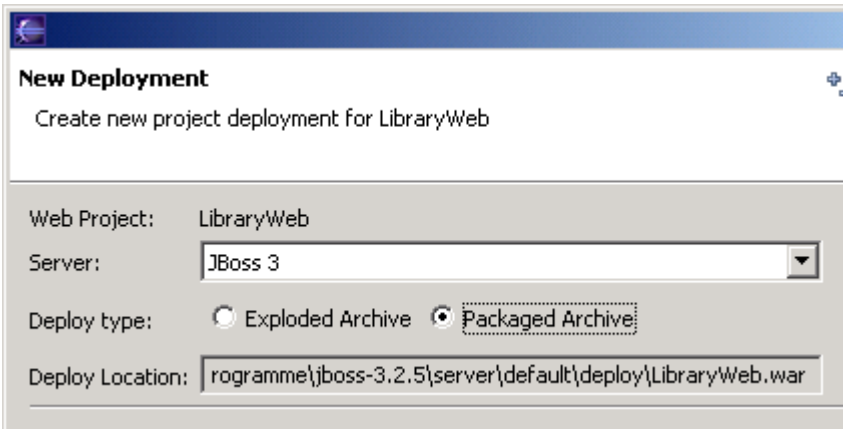
Create a new file named `editcustomer.jsp` in the folder `WebRoot/jsp/`.

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

```
<%@ 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"%>

<html>
      <head>
            <title>JSP for customerEditForm form</title>
      </head>
      <body>
            <html:form action="/customerEdit">
            <html:hidden property="id"/>
            <html:hidden property="do" value="saveCustomer"/>
            Name: <html:text property="name"/><br/>
            Last name <html:text property="lastname"/><br/>
            Age <html:text property="age"/><br/>
                  <html:submit/><html:cancel/>
            </html:form>
      </body>
</html>
```

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.

I hope you enjoyed the tutorial. If you have any feedback to us, feel free to contact us.