

1 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 basic tutorials for each technology first when you are a beginner.

Do you need expert help or consulting? Get it at <http://www.laliluna.de>

In-depth, detailed and easy-to-follow Tutorials for JSP, JavaServer Faces, Struts, Spring, Hibernate and EJB

Seminars and Education at reasonable prices on a wide range of Java Technologies, Design Patterns, and Enterprise Best Practices \implies Improve your development quality

An hour of support can save you a lot of time - Code and Design Reviews to insure that the best practices are being followed! \implies Reduce solving and testing time

Consulting on Java technologies \implies Get to know best suitable libraries and technologies

2 General

Author: Sebastian Hennebrueder

Date:

updated January, 2nd 2006

updated, July, 27th 2005

updated January, 16th 2005

First Edition December, 22th 2004

Used software and frameworks

Hibernate version 3.x (I used 3.1)

Struts 1.2

Java 5 or 1.5

Eclipse 3.x

MyEclipse 4 recommended but not obligatory

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

Database

I used PostgreSQL 8.0 and MySQL but you may use any database supported by Hibernate.

Application Server

Jboss 4.0.3 (our environment)

Tomcat standalone

probably any other application server like Jonas, WebSphere, Oracle, Websphere etc.

Source code: <http://www.laliluna.de/download/struts-hibernate-integration-tutorial.zip>

PDF version of the tutorial:

<http://www.laliluna.de/download/struts-hibernate-integration-tutorial-en.pdf>

Old version using Hibernate 2:

Table of content

| | | |
|-------|---|----|
| 1 | First steps using Struts and Hibernate..... | 1 |
| 2 | General..... | 2 |
| 3 | Requirements..... | 2 |
| 4 | Creating the application..... | 3 |
| 4.1 | Create the project and add the Hibernate capabilities (for MyEclipse user)..... | 3 |
| 4.2 | Preparing the project for anybody..... | 6 |
| 4.3 | Reduce Hibernate Libraries..... | 7 |
| 5 | Create the Database..... | 7 |
| 6 | Generate the Hibernate Mapping Files and Classes..... | 8 |
| 6.1 | Import using MyEclipse..... | 8 |
| 6.2 | Repair the mapping of customer..... | 10 |
| 6.3 | Repair the mapping of book..... | 12 |
| 6.3.1 | Correct the Boolean mapping..... | 12 |
| 6.4 | Improvements to the session factory..... | 13 |
| 6.5 | Testing the Hibernate part..... | 13 |
| 6.6 | PostgreSQL Problem..... | 15 |
| 7 | Generating the Business Logic..... | 16 |
| 7.1 | Create a business logic class..... | 16 |
| 8 | Creating the dialogs with Struts..... | 25 |
| 8.1 | Create a default, welcome page..... | 26 |
| 8.2 | Global Action Forwards and Action Mappings..... | 27 |
| 8.3 | Book list..... | 30 |
| 8.3.1 | Action mapping und action class of the book list..... | 32 |
| 8.3.2 | Edit the source code of the action form class..... | 33 |
| 8.3.3 | Edit the source code of the action class..... | 33 |
| 8.3.4 | Display the books list in the jsp file..... | 34 |
| 9 | Test the application..... | 37 |
| 9.1 | Add, edit, borrow and delete books..... | 37 |
| 9.1.1 | Action Mapping | 37 |
| 9.1.2 | Edit the source code of the jsp files..... | 40 |
| 9.1.3 | Form bean | 42 |
| 9.1.4 | Methods of the dispatch action class..... | 46 |
| 9.2 | Use case Customer list..... | 49 |
| 9.2.1 | Edit the source code of the action form class..... | 52 |
| 9.2.2 | Displaying the custom list..... | 53 |
| 9.3 | Use case add, edit, delete customers..... | 54 |
| 9.3.2 | Customer form bean..... | 57 |
| 9.3.3 | Edit the source code of the action class..... | 59 |
| 9.3.4 | Edit the source code of the jsp file..... | 60 |
| 10 | Test the applications..... | 62 |
| 11 | Copyright and disclaimer..... | 62 |

3 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 we put everywhere.

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

If you want to use free tools for web application development, have a look at the tutorial <http://www.laliluna.de/first-steps-with-struts-free-tools-en.html>

4 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.

4.1 Create the project and add the Hibernate capabilities (for MyEclipse user)

Create a new web project.

So let's start.

Press *Ctrl+n* (or *Strg+n*) to open the „New ...“ dialog.

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

New J2EE Web Project

Create web project



Web Project Details

Project Name

Location Use default location

Directory

Source folder

Web root folder

Context root URL

J2EE Specification Level

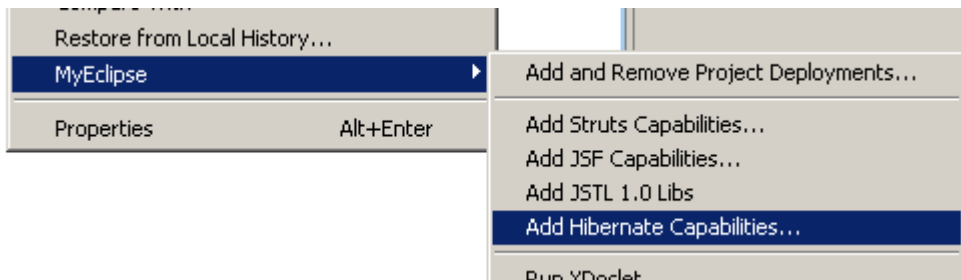
J2EE 1.3 J2EE 1.4 [default]

JSTL Support

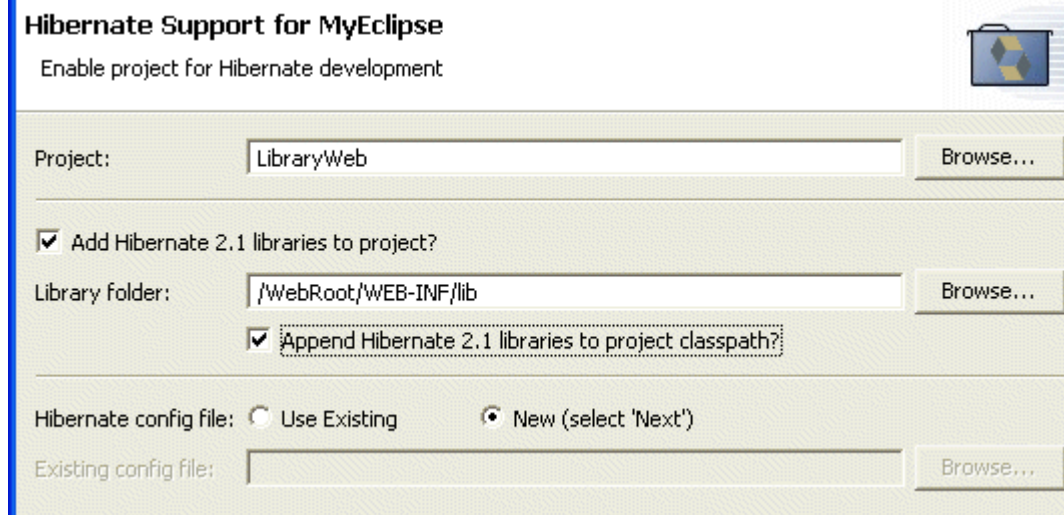
Add JSTL libraries to WEB-INF/lib folder?

JSTL 1.0 JSTL 1.1

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.

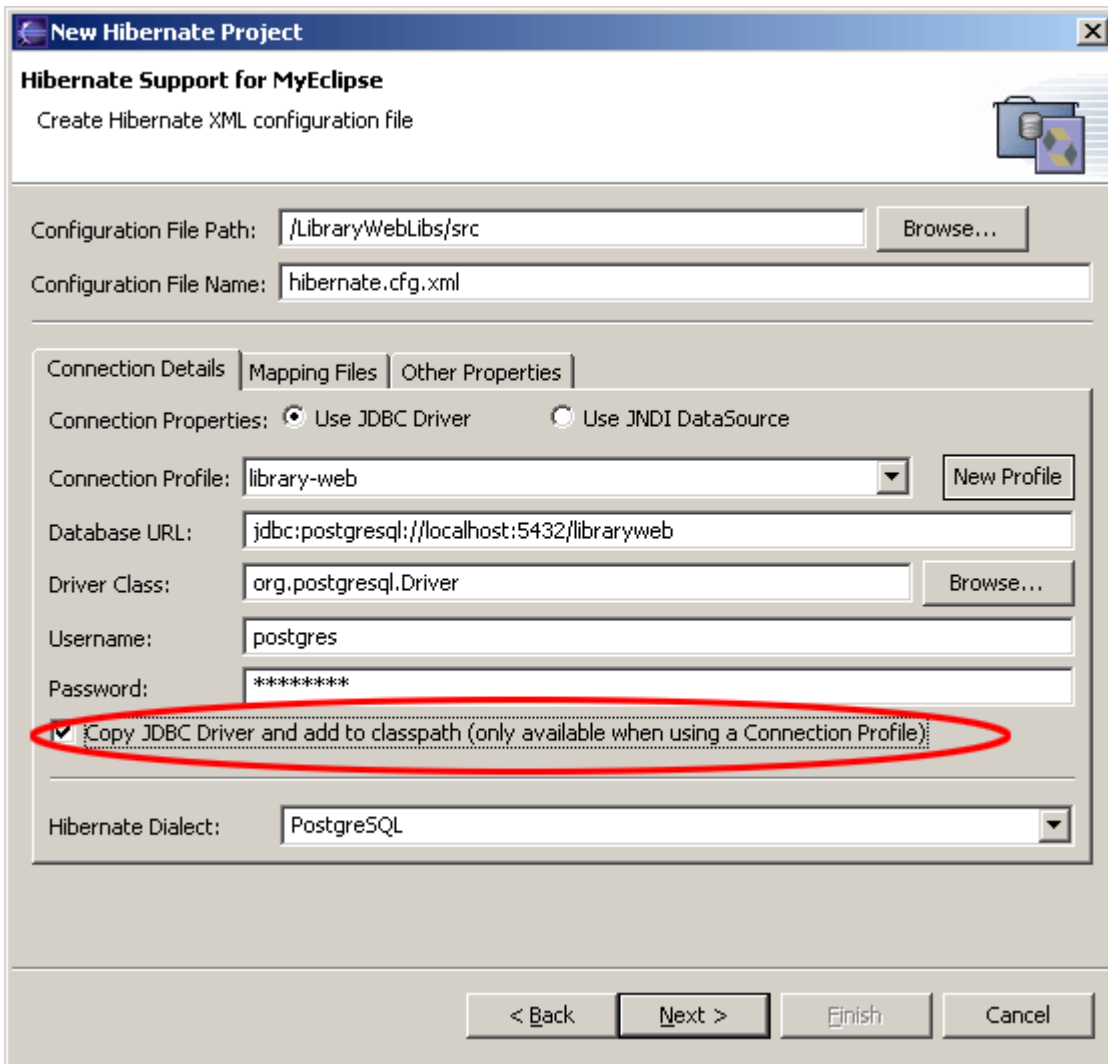
When the Postgre Driver is missing. Click on „New Driver“ to create a new driver. You will need the jar including your Database Driver.

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

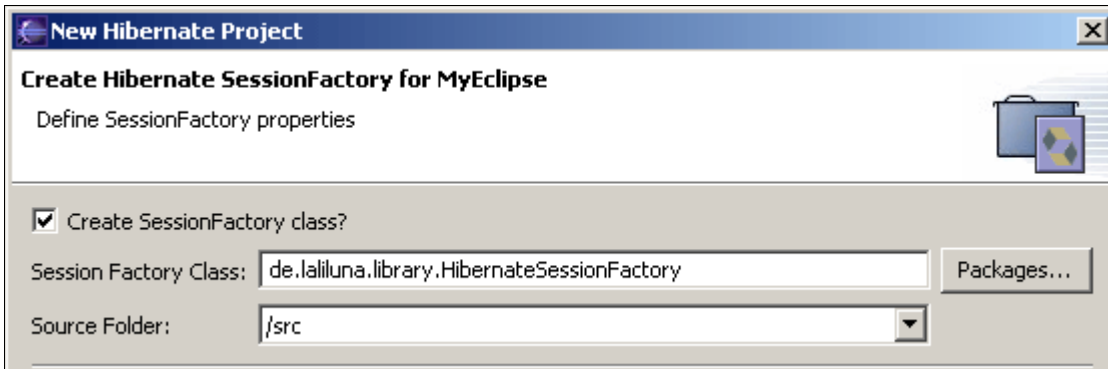
New Profile



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 source code you will find also a configuration for MySQL.



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 does what a factory does normally do ;-).

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 pattern. 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 we provided.

4.2 Preparing the project for anybody

MyEclipse provides functionality to create and deploy web projects to a wide choice of application server. Create a web project as explained in the Struts free tools tutorial, we mentioned at the beginning.

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 3 to work. 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.

- + xml-apis.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + asm.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + asm-attrs.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + c3p0-0.9.0.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + cglib-2.1.3.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + commons-collections-2.1.1.jar - C:\setting\java\libs\hibernate-3.1-mini
- + commons-logging-1.0.4.jar - C:\setting\java\libs\hibernate-3.1-minima
- + concurrent-1.3.2.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + dom4j-1.6.1.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + ehcache-1.1.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + hibernate3.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + jacc-1_0-fr.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + jaxen-1.1-beta-7.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + jdbc2_0-stdext.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + jta.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + log4j-1.2.11.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + antlr-2.7.6rc1.jar - C:\setting\java\libs\hibernate-3.1-minimal
- + JAVA_LIBS/postgresql-8.0-312.jdbc3.jar - C:\setting\java\libs\postgresql-

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. In the source code you can find the configuration for a MySQL database, I use PostgreSQL here.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

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

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

That's all for the non MyEclipse users.

4.3 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.

5 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,
  firstname 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,
  borrowallowed bool NOT NULL DEFAULT true,
  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 ,
  firstname 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 ) ,
borrowallowed 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 ;
```

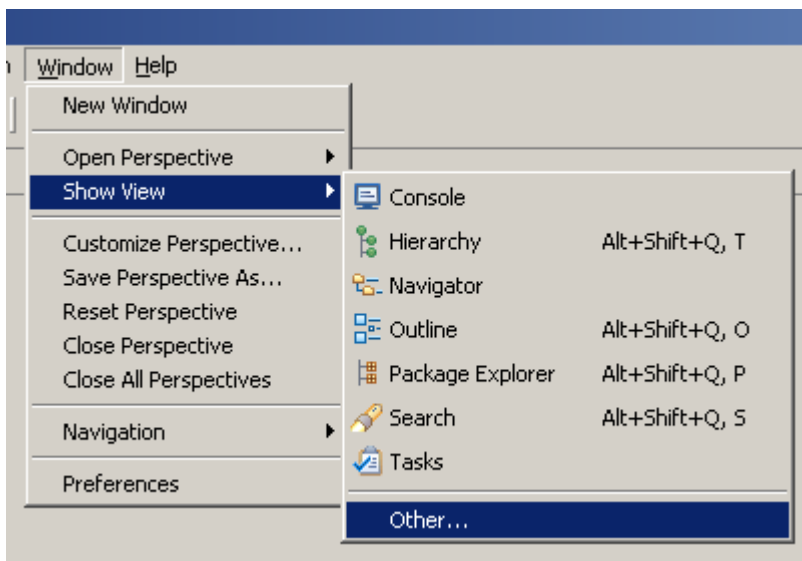
6 Generate the Hibernate Mapping Files and Classes

6.1 Import using MyEclipse

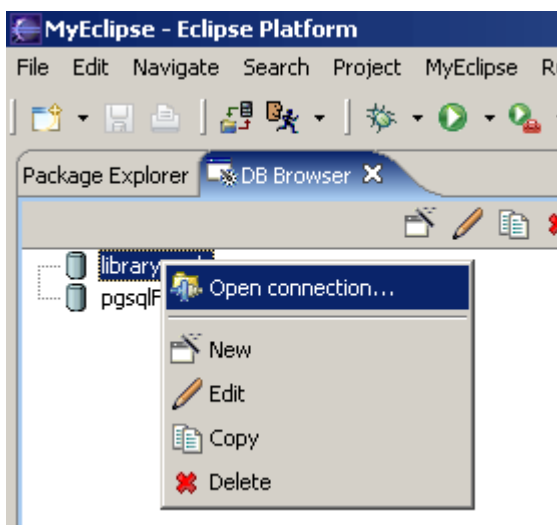
Non MyEclipse user

We will import the tables using MyEclipse to generate the raw mappings. You may use the Hibernate tools to do the same thing. You can find all mappings below.

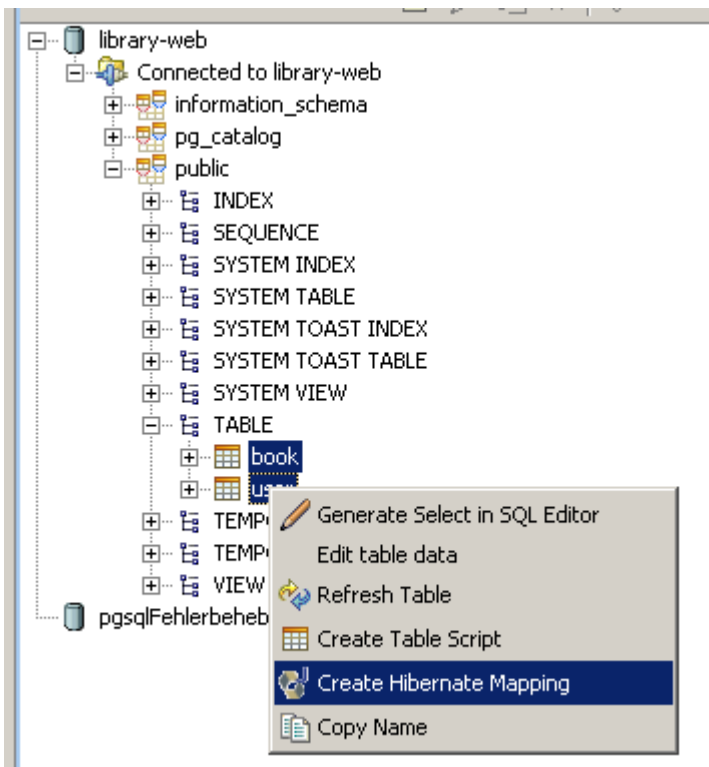
Open the View DB Browser (MyEclipse). If you cannot find it open the „Show View“ Dialog and select in the MyEclipse Enterprise Workbench the DB Browser.



Open the connection profile you specified before.



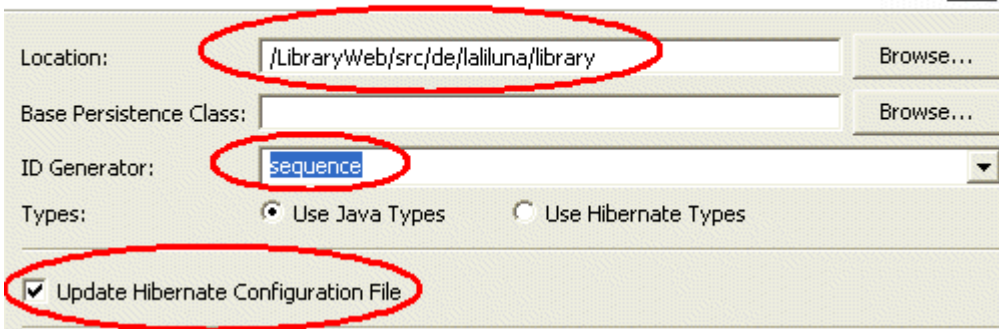
Select the two tables we have just created. Right click and choose „Create Hibernate Mapping“.



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

Hibernate Mapping Generation

Generate Hibernate Mapping and Java Class for Database Explorer Table



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. Your foreign key may or may not have been generated. This function is relative new and not working on all platforms.

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
           "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
           "http://hibernate.sourceforge.net/hibernate-mapping-
3.0.dtd" >
<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="customerFk" column="customer_fk" type="java.lang.Integer" />
    <property name="borrowallowed" column="borrowallowed" type="java.lang.Byte" />
  </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="increment"/>
  </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.

6.2 Improve the mapping of customer

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 Set books;
return customer;
}

public Set getBooks() {
return books;
```

```

}

public void setBooks(Set books) {
    this.books = books;
}

```

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

```

<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="de.laliluna.library">
<class name="Customer" table="customer">
    <id name="id" column="id" type="java.lang.Integer">
        <generator class="sequence">
            <param name="sequence">customer_id_seq</param>
        </generator>
    </id>
    <set name="books" inverse="false" >
        <key>
            <column name="customer_fk"></column>
        </key>
        <one-to-many class="Book" />
    </set>
    <property name="firstname" column="firstname" 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>

```

```

<generator class="sequence">

```

This tag specifies how the id is generated. Using PostgreSQL the sequence `customer_id_seq` is called. The generator depends on your database. For MySQL you will find the example in the [source](#).

```

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

```

We have a set, which is accessed by the name `books`, i.e. `GetBooks` and `setBooks` in the `Customer` class. The foreign key column is `customer_fk` and related to the mapping of the class `Book`.

```

inverse="false"

```

This is a little more complex to explain. You can write a relation from two sides.

```

customer.getBooks().add(book);

```

or

```
book.setCustomer(customer);
```

Very often you only need to set the relation from one side. So you can inform Hibernate that it needs not to monitor the offer side of the relation. This could be a performance issue. In this case we would set `inverse="true"` and could only use the second approach to write a relation.

When you must use `inverse="true"`?

Imagine a 1:n relation author to book. A book must have at least one author. The foreign key column `author_fk` cannot be null opposed to our tutorial example, where the column `customer_fk` can be null when the book is not borrowed.

When you use not `inverse true` than there is a possibility that the book is written to the database with the `author_fk` set to null. In this case you must set the author always on the book side.

```
book.getAuthors().add(author);
```

In the class `customer` I overwrote the `toString` method to have a proper output during debugging. In addition I changed the constructor method to have the property `books` initialized with an empty `hashset`. The advantage is that you do not have to test in the business logic if your `hashSet` is null or empty.

```
package de.laliluna.library;

import java.io.Serializable;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

/**
 * A class that represents a row in the 'customer' table. This class may be
 * customized as it is never re-generated after being created.
 */
public class Customer extends AbstractCustomer implements Serializable {

    private Set books;

    /**
     * Simple constructor of Customer instances.
     */
    public Customer() {
        books = new HashSet();
    }

    /**
     * Constructor of Customer instances given a simple primary key.
     *
     * @param id
     */
    public Customer(java.lang.Integer id) {
        super(id);
        books = new HashSet();
    }

    public Set getBooks() {
        return books;
    }

    public void setBooks(Set books) {
        this.books = books;
    }

    /* Add customized code below */
    @Override
    public String toString() {
```

```

return "Customer " + getId() + " firstname: " + getFirstname()
    + " lastname: " + getLastname();
}
}

```

6.3 Improve the mapping of book

See below the content of the Book.hbm.xml file. When you imported this with MyEclipse please correct the borrowAllowed property. The postgres bool column is recognized as Byte and as Short when you are using MySQL. Do not ask me why.

```

<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="de.laliluna.library">
  <class name="Book" table="book">
    <id name="id" column="id" type="java.lang.Integer">
      <generator class="sequence">
        <param name="sequence">book_id_seq</param>
      </generator>
    </id>
    <property name="title" column="title" type="java.lang.String" />
    <property name="author" column="author" type="java.lang.String" />
    <property name="borrowallowed" column="borrowallowed" type="java.lang.Boolean"/>
    <many-to-one name="customer" column="customer_fk" not-null="false" >
      </many-to-one>
    </class>
  </hibernate-mapping>

```

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 files due to use of tabs in this document. If you encounter funny problems when pasting content to Eclipse, please consider to type the code manually.

6.3.1 Correct the Boolean mapping

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

```

/** The value of the simple borrowallowed property. */
private java.lang.Boolean borrowallowed;
/**
 * Return the value of the borrowallowed column.
 * @return java.lang.Byte
 */
public java.lang.Boolean getBorrowallowed()
{
    return this.borrowallowed;
}

```

```

    }

    /**
     * Set the value of the borrowallowed column.
     * @param borrowallowed
     */
    public void setBorrowallowed(java.lang.Boolean borrowallowed)
    {
        this.borrowallowed = borrowallowed;
    }

```

In the class `Book` I overwrote the `toString` method to get a proper output during debugging. If the relation to customer is not detected properly, here is the complete code of the class.

```

package de.laliluna.library;

import java.io.Serializable;

/**
 * A class that represents a row in the 'book' table.
 * This class may be customized as it is never re-generated
 * after being created.
 */
public class Book
    extends AbstractBook
    implements Serializable
{
    private Customer customer;

    @Override
    public String toString() {
        return "Book: " + getId()+" title: "+getTitle()+" author: "+getAuthor() + "
borrowed to: "+getCustomer();
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    /**
     * Simple constructor of Book instances.
     */
    public Book()
    {
    }

    /**
     * Constructor of Book instances given a simple primary key.
     * @param id
     */
    public Book(java.lang.Integer id)
    {
        super(id);
    }

    /* Add customized code below */
}

```

6.4 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;
}
```

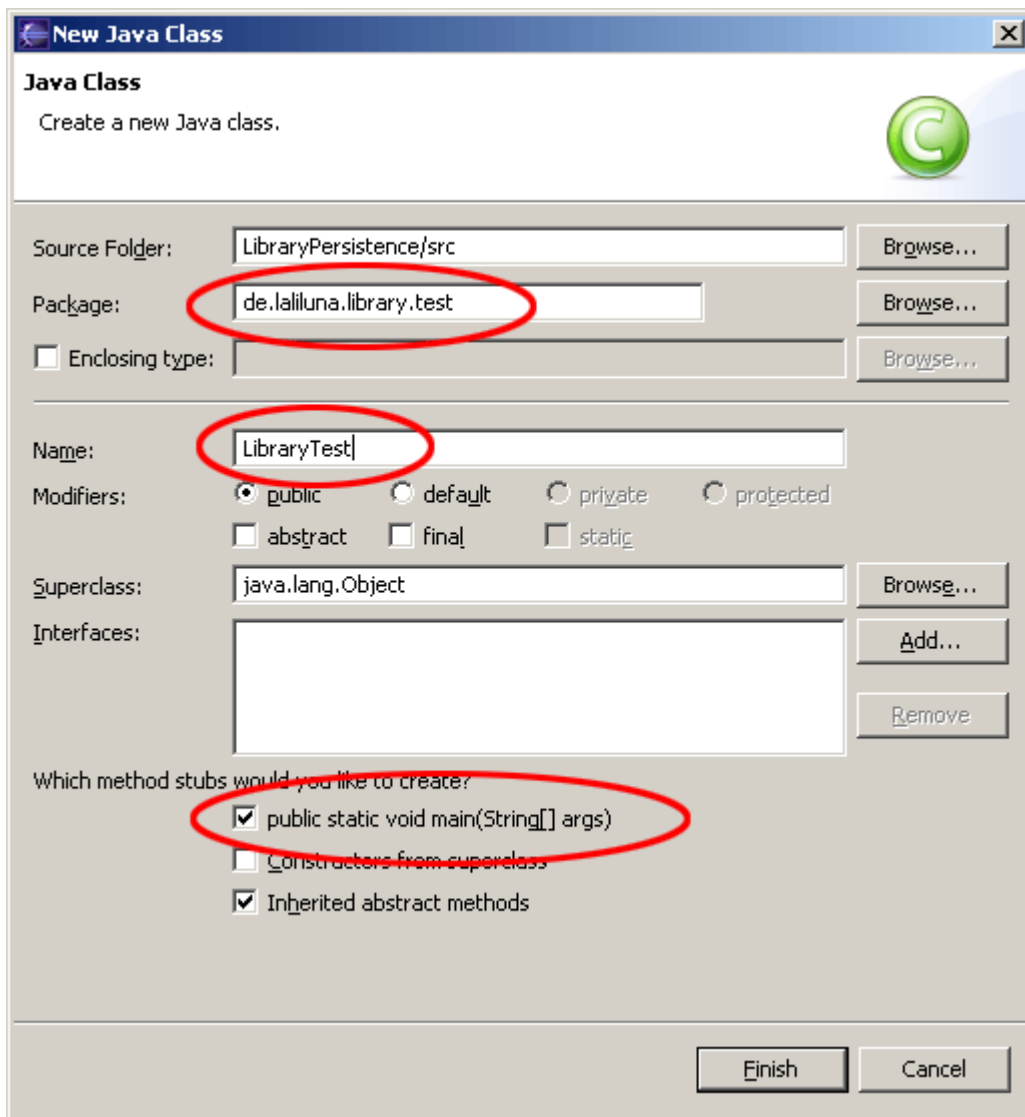
6.5 Testing the Hibernate part

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. You can find a basic `log4j.properties` file in the source.

Create a new class to implement the test methods.



Add the following content:

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

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

import org.apache.log4j.Logger;
import org.hibernate.Session;
import org.hibernate.Transaction;

import de.laliluna.library.Book;
import de.laliluna.library.Customer;
import de.laliluna.library.HibernateSessionFactory;

public class LibraryTest {
    private Session session;

    private Logger log;

    public static void main(String[] args) {
```



```

/*
 * hibernate needs log4j. Either create a log4j.properties file in the
 * source directory * or alternatively make the following to create a
 * standard configuration BasicConfigurator.configure();
 */
LibraryTest libraryTest = new LibraryTest();
try {
    libraryTest.setUp();
    libraryTest.testCreateDomains();
    libraryTest.testAddRemoveRelation();
    libraryTest.listBooks();
    libraryTest.tearDown();
} catch (Exception e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

private void testCreateDomains() {
    Transaction tx = session.beginTransaction();
    Book book = new Book();
    book.setAuthor("Sebastian");
    book.setTitle("Hibernation in winter");
    book.setBorrowallowed(true);
    session.save(book);
    tx.commit();
    tx = session.beginTransaction();
    Customer customer = new Customer();
    customer.setLastname("Liebzeit");
    customer.setFirstname("Carsten");
    customer.setAge(25);
    session.save(customer);
    tx.commit();
}

private void testAddRemoveRelation() {
    log.info("Adding and removing relations");
    Transaction tx = session.beginTransaction();

    // create two books and a customer
    Book book = new Book();
    book.setAuthor("Sebastian's");
    book.setTitle("Hibernation in the summer");
    book.setBorrowallowed(true);
    session.save(book);
    Book book2 = new Book();
    book2.setAuthor("Karl May");
    book2.setTitle("Wildes Kurdistan");
    book2.setBorrowallowed(true);
    session.save(book2);

    Customer customer = new Customer();
    customer.setLastname("Meier");
    customer.setFirstname("John");

```

```

customer.setAge(25);
session.save(customer);

//customer borrows books
customer.getBooks().add(book);
customer.getBooks().add(book2);
session.flush();

session.refresh(customer);
session.refresh(book);
session.refresh(book2);
Set books = customer.getBooks();
log.info("list books of customer");
for (Iterator iter = books.iterator(); iter.hasNext();) {
    Book element = (Book) iter.next();
    log.info(element);
}
//first book is returned
book.setCustomer(null);
customer.getBooks().remove(book);
session.flush();
session.refresh(customer);
log.info("list books of customer");
books = customer.getBooks();
for (Iterator iter = books.iterator(); iter.hasNext();) {
    Book element = (Book) iter.next();
    log.info(element);
}

tx.commit();
tx = session.beginTransaction();
session.delete(customer);
session.delete(book);
session.delete(book2);
tx.commit();
}

protected void setUp() throws Exception {
    session = HibernateSessionFactory.currentSession();
    log = Logger.getLogger(this.getClass());
}

protected void tearDown() throws Exception {
    HibernateSessionFactory.closeSession();
}

/**
 * creates a book and saves it to the db.
 *
 */

/**
 * lists all books in the db
 *
 */

```

```

*/
private void listBooks() {
    log.info("##### list customers");

    Transaction tx = session.beginTransaction();
    List customers = session.createQuery("select c from Customer as c")
        .list();
    for (Iterator iter = customers.iterator(); iter.hasNext();) {
        Customer element = (Customer) iter.next();
        Set books = element.getBooks();
        System.out.println(element);
        if (books == null)
            log.info("no books");
        else {
            for (Iterator iterator = books.iterator(); iterator.hasNext();) {
                Book book = (Book) iterator.next();
                log.info(" - " + book);
            }
        }
        log.info(element);
    }
    tx.commit();

    log.info("##### list books");

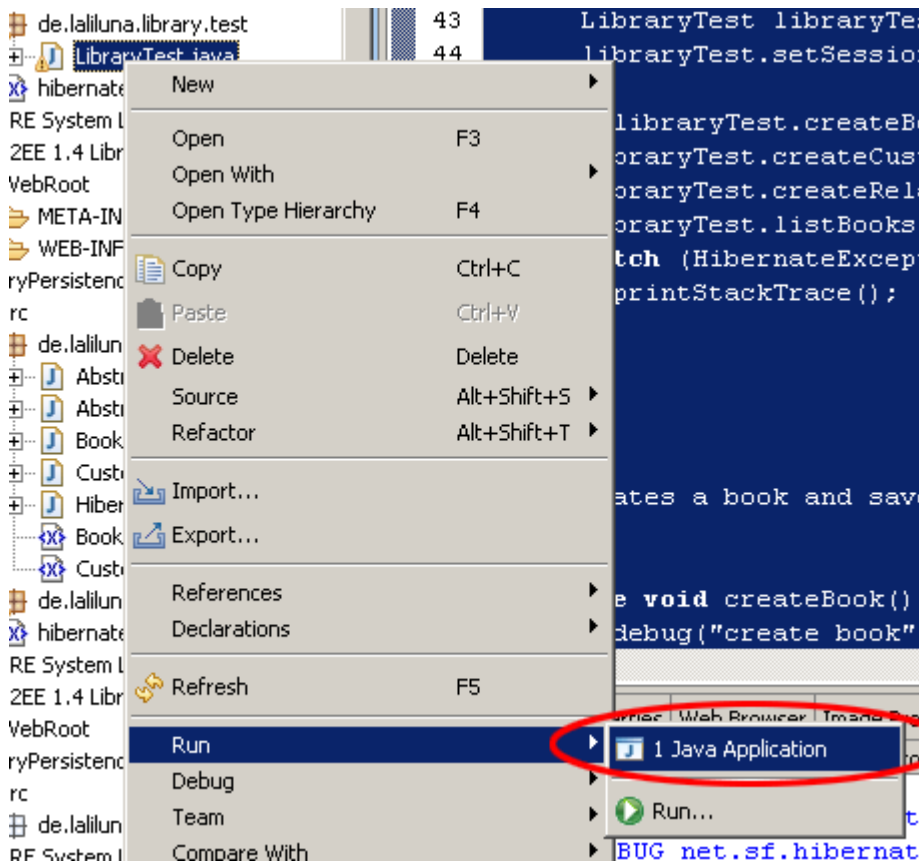
    tx = session.beginTransaction();
    List books = session.createQuery("select b from Book as b").list();
    for (Iterator iter = books.iterator(); iter.hasNext();) {
        System.out.println((Book) iter.next());
    }
    tx.commit();
}

/**
 * @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

6.6 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.

```
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">
```

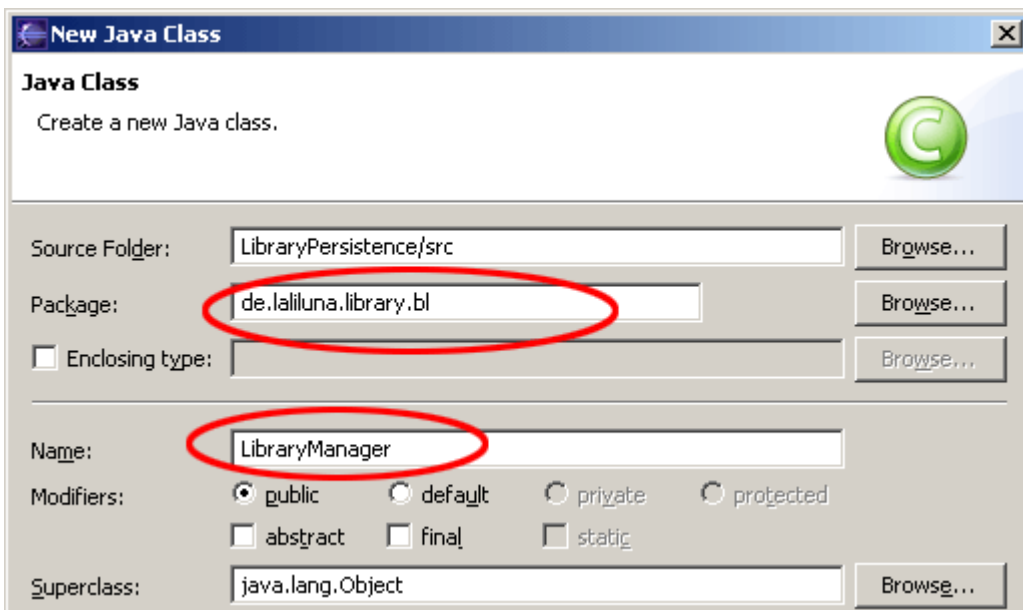
```
<param name="sequence">book_id_seq</param>
</generator>
```

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

7 Generating the Business Logic

7.1 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 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 java.util.Set;

import org.hibernate.Session;
import org.hibernate.Transaction;

import de.laliluna.library.Book;
import de.laliluna.library.Customer;
import de.laliluna.library.HibernateSessionFactory;

/**
 * @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;

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

    tx = session.beginTransaction();
    List tmpBooks = session.createQuery(
        "select b from Book as b order by b.author, b.title").list();
    for (Iterator iter = tmpBooks.iterator(); iter.hasNext();) {
        books.add((Book) iter.next());
    }
    tx.commit();

    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;

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

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

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

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

    tx = session.beginTransaction();
    Book book = (Book) session.get(Book.class, primaryKey);
    // only if borrowing of the book is allowed
    if (book.getBorrowallowed()) {
        Customer customer = (Customer) session.get(Customer.class,
            customerPrimarykey);
        if (book != null && customer != null)
            book.setCustomer(customer);
    }

    tx.commit();
}

/**
 * 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;
/* 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
    Customer customer = book.getCustomer();
    if (customer != null) {
        customer.getBooks().remove(book);
        book.setCustomer(null);
    }
}
tx.commit();
}

/**
 * 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;
    /* 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.setBorrowallowed(bookValue.getBorrowallowed());
            session.update(book);
        }
    } else // [laliluna] 04.12.2004 create new book
    {
        book = new Book();
        book.setAuthor(bookValue.getAuthor());
        book.setTitle(bookValue.getTitle());
        book.setBorrowallowed(bookValue.getBorrowallowed());
        session.save(book);
    }
}

```



```

    tx.commit();
}

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

    /* get session of the current thread */
    session = HibernateSessionFactory.currentSession();
    tx = session.beginTransaction();
    Book book = (Book) session.get(Book.class, primaryKey);
    if (book != null)
        if (book.getCustomer() != null)
            book.getCustomer().getBooks().remove(book);
    session.delete(book);
    tx.commit();
}

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

    tx = session.beginTransaction();
    List tmpCustomer = session.createQuery(
        "select c from Customer as c order by c.lastname").list();
    for (Iterator iter = tmpCustomer.iterator(); iter.hasNext();) {
        customers.add((Customer) iter.next());
    }
    tx.commit();
    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;

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

    tx = session.beginTransaction();
    customer = (Customer) session.get(Customer.class, primaryKey);
    tx.commit();

    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;

    /* 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.setFirstname(customer.getFirstname());
        session.update(toBeUpdated);
    }
    tx.commit();
}

/**
 * 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;

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

    tx = session.beginTransaction();
    Customer customer = (Customer) session.get(Customer.class, primaryKey);
    if (customer != null) {
        Set books = customer.getBooks();
        for (Iterator iter = books.iterator(); iter.hasNext();) {
            Book element = (Book) iter.next();
            element.setCustomer(null);
        }
        session.delete(customer);
    }
    tx.commit();
}
}

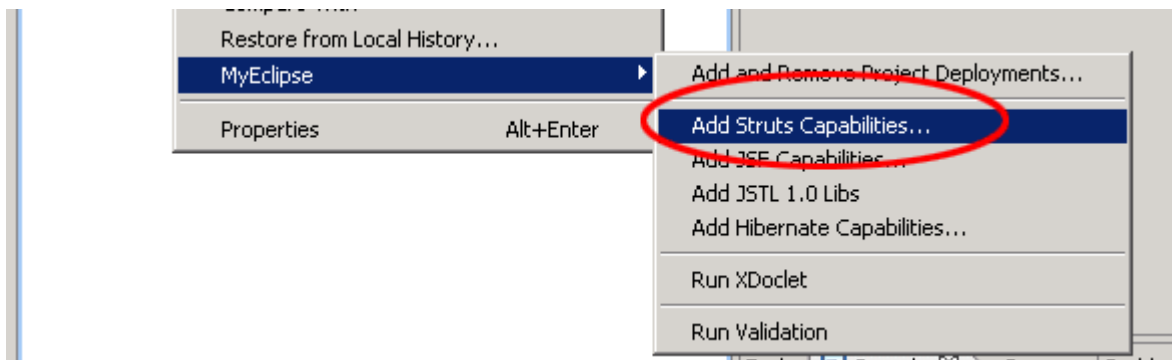
```

That's all we have created our business logic.

And now the last part: the dialogs

8 Creating the dialogs with Struts

For now your project is still a normal Web project, so we need to add the struts capabilities. Right click on the project and add the capabilities for struts with Add Struts Capabilities.



Change the **Base package** for new classes and the **Default application resource**.

Struts Support for MyEclipse Web Project

Enable project for Struts development



| | |
|-------------------------------|--|
| Web project: | LibraryWeb |
| Web-root folder: | /WebRoot |
| Servlet specification: | 2.4 |
| Struts config path: | <input type="text" value="/WEB-INF/struts-config.xml"/> <input type="button" value="Browse..."/> |
| Struts specification: | <input type="radio"/> Struts 1.0 <input type="radio"/> Struts 1.1 <input checked="" type="radio"/> Struts 1.2 |
| ActionServlet name: | <input type="text" value="action"/> |
| URL pattern | <input checked="" type="radio"/> *.do <input type="radio"/> /do/* |
| Base package for new classes: | <input type="text" value="de.laliluna.library.struts"/> <input type="button" value="Browse..."/> |
| Default application resource: | <input type="text" value="de.laliluna.library.struts.ApplicationResources"/> |
| | <input checked="" type="checkbox"/> Install Struts jars <input checked="" type="checkbox"/> Install Struts TLDs |

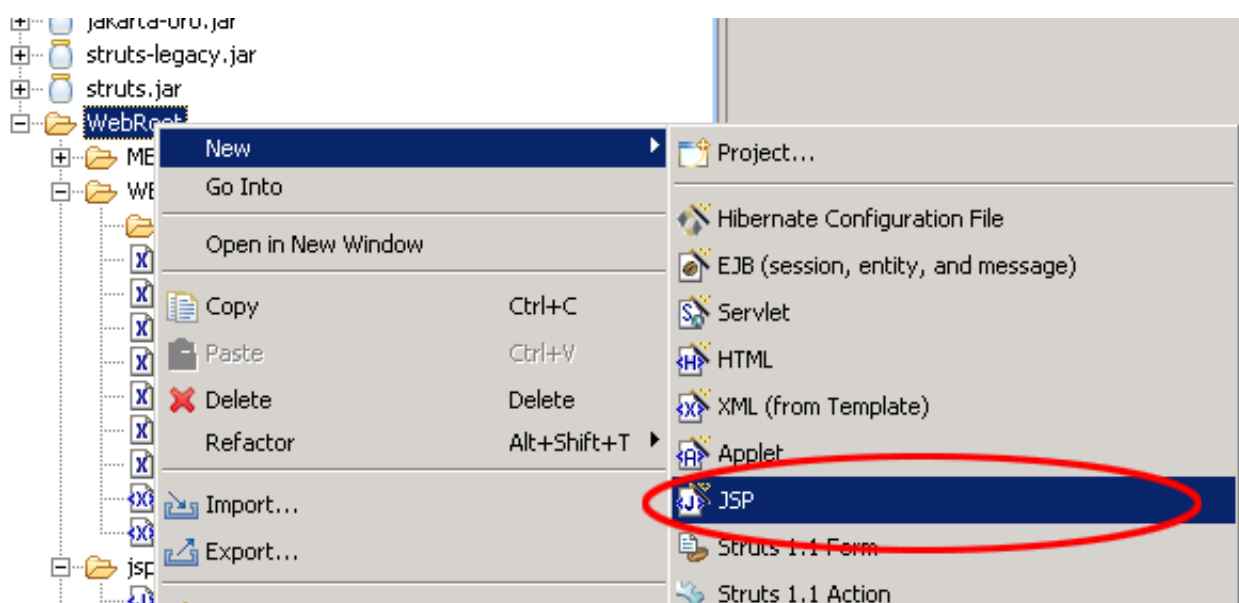
Notice

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.

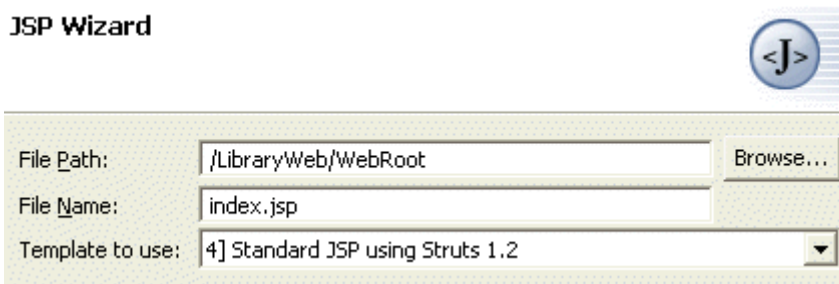
8.1 Create a default, welcome page

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

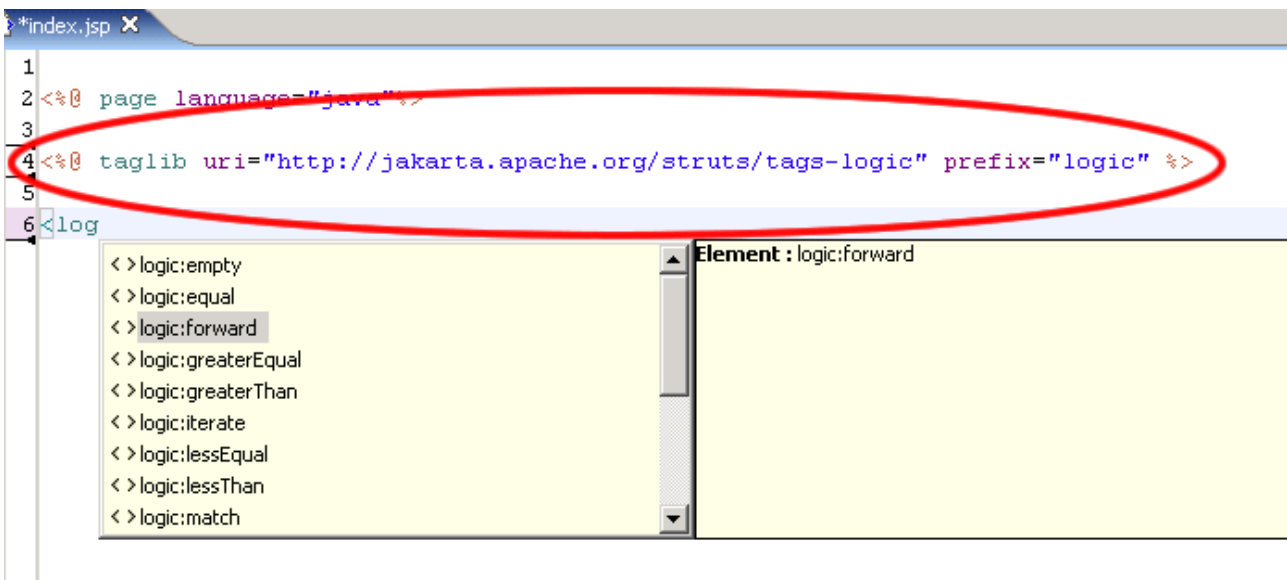


Set the name to `index.jsp` and choose on template to use `> Standard JSP using Struts 1.2` MyEclipse will use the template to create the JSP File.

JSP Wizard



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>
```

8.2 Global Action Forwards and Action Mappings

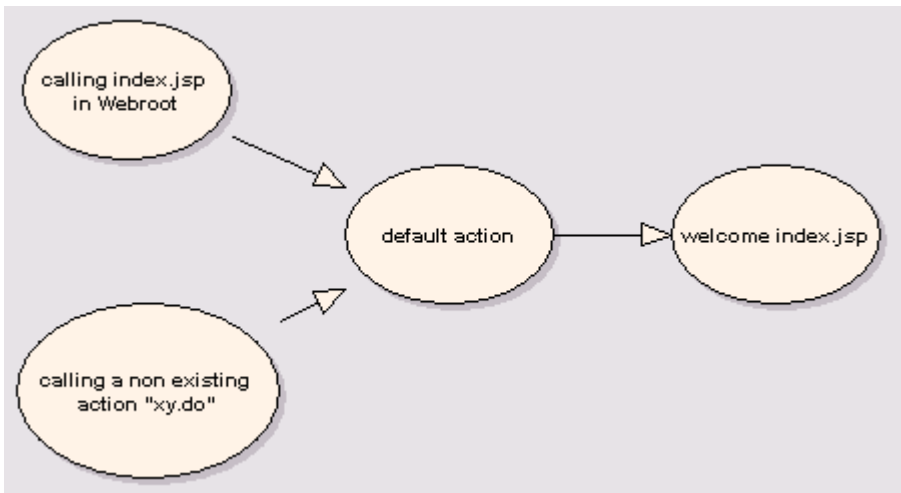
What is an action forward?

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.

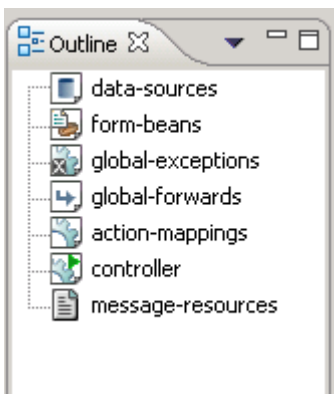
What is a action mapping?

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.



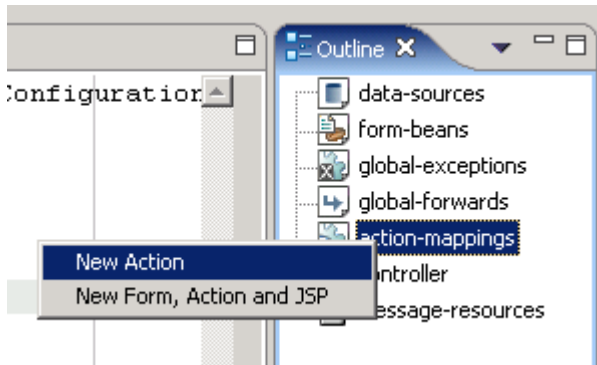
In the first step we create a new action mapping. Open the `struts-config.xml`, you will find it in the folder `WebRoot/WEB-INF`. Right click in the outline view on `action-mapping`.



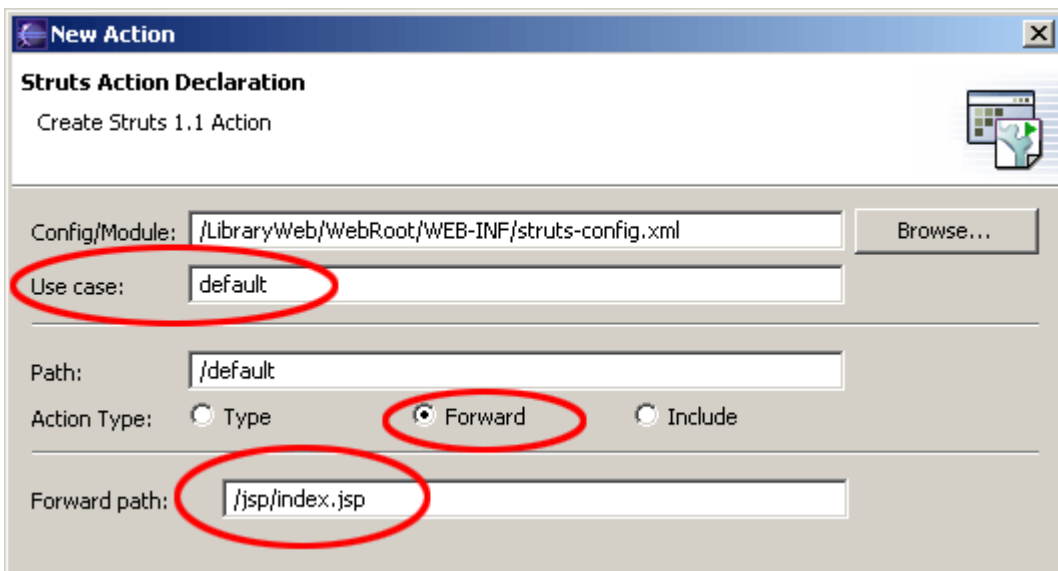
MyEclipse provides some nice features for creating struts files. Open the `struts-config.xml` and the

Outline View.

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



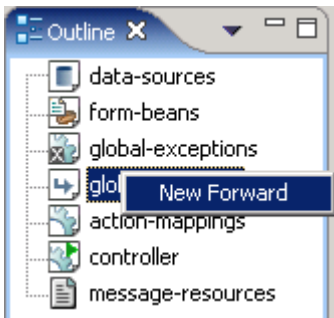
Choose **Use Case** default and **Action Type** Forward. The **Forward Path** is the welcome page /jsp/index.jsp



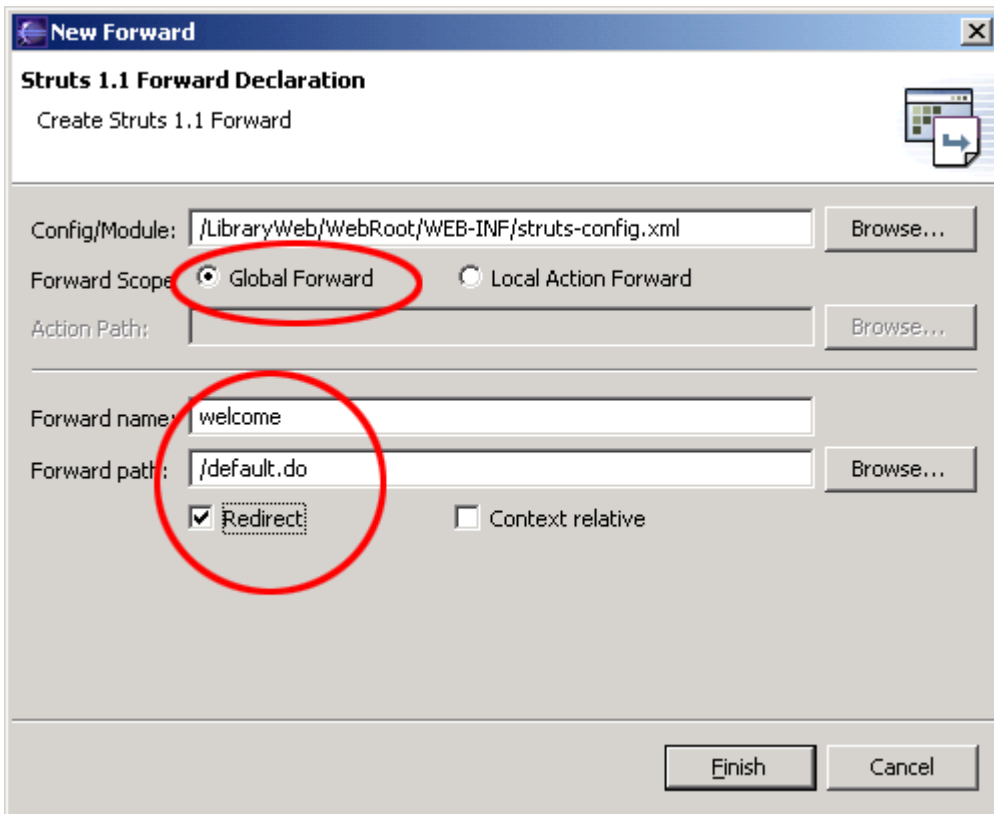
To catch all requests of non existing action mappings, we have to add manually a parameter `unknown="true"` to the action forward.

```
<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>
</struts-config>
```



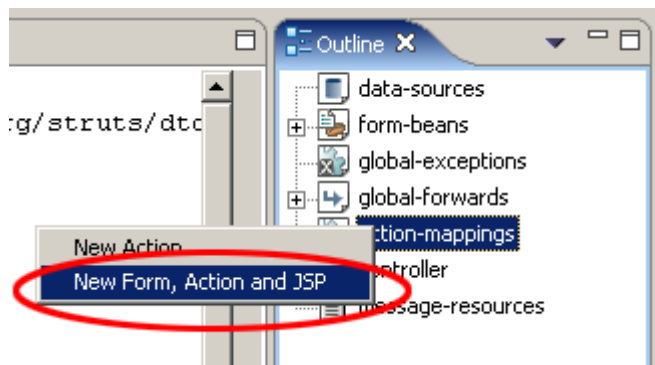
```
<action-mappings >
  <action forward="/jsp/index.jsp" path="/default" unknown="true" />
</action-mappings>

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

8.3 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.

New Form

Struts 1.1 Form Declaration
Create Struts 1.1 FormBean

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

Use case: bookList

Name: bookListForm

Form Impl: New FormBean Existing FormBean Dynamic FormBean

Superclass: org.apache.struts.action.ActionForm

Form type: de.laliluna.tutorial.library.struts.form.BookListForm

Optional Details

Form Properties **Methods** JSP

Create methods?

- public ActionErrors validate(HttpServletRequest...)
- public void reset(HttpServletRequest...)
- public ActionErrors validate(ServletRequest...)
- public void reset(ServletRequest...)

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

Optional Details

Form Properties Methods **JSP**

Create JSP form?

New JSP Path: /jsp/bookList.jsp

Press the next button to continue to the action mapping.

8.3.1 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`

Config/Module:

Path:

Action Type: Type Forward Include

Action Impl: Create new Action class Use existing Action class

Superclass:

New Type:

Optional Details

Form | Parameter | Methods | Forwards | Exceptions

Name:

Attribute:

Scope:

Validate Form

Input Source:

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

New Forward

Forward name:

Forward path:

Redirect Context relative

That's it. Let the files be generated.

Notice

MyEclipse adds the action mapping to the `struts-config`, creates a blank 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>
```

8.3.2 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.

8.3.3 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;
    // [laliluna] 27.11.2004 get busines logic
    LibraryManager libraryManager = new LibraryManager();
    // [laliluna] 29.11.2004 update the form bean, from which the jsp will read the
    data later.
    bookListForm.setBooks(libraryManager.getAllBooks());
    return mapping.findForward("showList");
}
}

```

8.3.4 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>Borrowing allowed</td>
    <td>Borrow by</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td>&nbsp;</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="borrowallowed"/>
</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.firstname" />,
        <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 user 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>
&nbsp;
<html:button property="back"
             onclick="location.href='default.do'">Back to menu
</html:button>

```

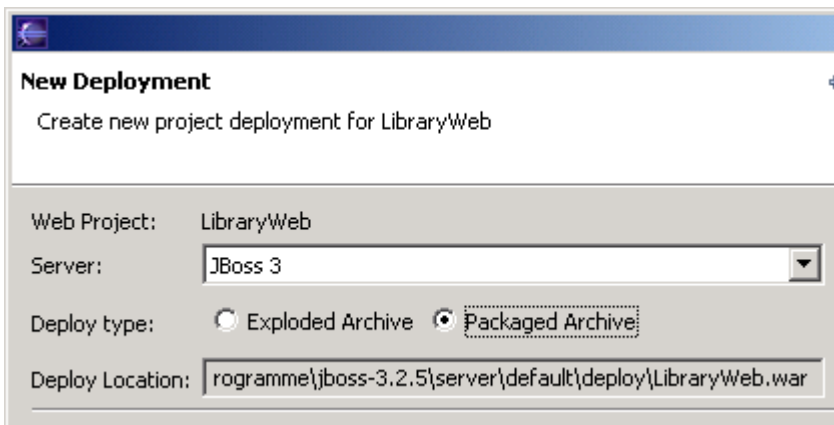
```
</body>
</html>
```

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.

9 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. Filejar 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.

9.1 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

9.1.1 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.

The screenshot shows the 'Form' tab of the Struts configuration wizard. The 'Config/Module' field is set to '/LibraryWeb/WebRoot/WEB-INF/struts-config.xml'. The 'Use case' field is 'bookEdit'. The 'Path' is '/bookEdit'. The 'Action Type' is 'Type'. The 'Action Impl' is 'Create new Action class'. The 'Superclass' is 'org.apache.struts.actions.DispatchAction'. The 'New Type' is 'de.laliluna.tutorial.library.struts.action.BookEditAction'. The 'Optional Details' section has tabs for 'Form', 'Parameter', 'Methods', 'Forwards', and 'Exceptions'. The 'Form' tab is active, showing 'Name' as 'bookEditForm', 'Attribute' as 'bookEditForm', 'Scope' as 'request', and 'Validate Form' checked. There are 'Browse...' buttons next to the 'Name' and 'Input Source' fields.

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

The screenshot shows the 'Parameter' tab of the Struts configuration wizard. The 'Parameter' field is 'do'.

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.

Forward name: showEdit

Forward path: /jsp/bookEdit.jsp

Redirect Context relative

Add Close Browse...

Forward name: showAdd

Forward path: /jsp/bookAdd.jsp

Redirect Context relative

Add Close Browse...

Forward name: showBorrow

Forward path: /jsp/borrowBook.jsp

Redirect Context relative

Update Close Browse...

Forward name: showList

Forward path: /bookList.do

Redirect Context relative

Add Close Browse...

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

9.1.2 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>
      </tbody>
    </table>
  </form>

```

```

        </td>
    </tr>
    <tr>
        <td>
            Borrow allowed:
        </td>
        <td>
            <html:checkbox property="borrowallowed" />
        </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>
        &nbsp;
        <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 be called.

Open the file **bookEdit.jsp**. You can use the source code 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>
          First name
        </td>
        <td>
          Borrow
        </td>
      </tr>

      <!-- start with an iterate over the collection users -->
      <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="firstname" />
            </td>
            <td>
              <html:radio property="customerId" value="{customer.id}" />
            </td>
          </tr>
        </logic:iterate>
      </logic:present>
      <!-- end iterate -->

      <!-- 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>

```

9.1.3 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 them 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 getBorrowallowed() {
    return book.getBorrowallowed();
}

```

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.

```

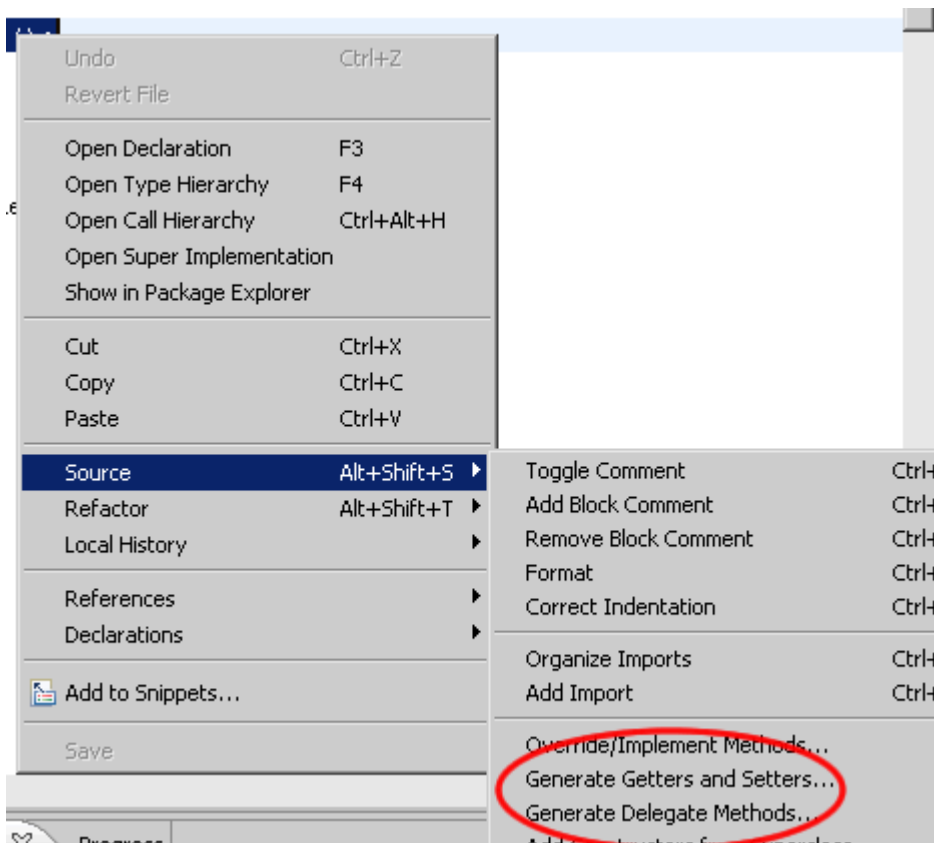
public class BookEditForm extends ActionForm {

    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();
    book.setBorrowallowed(false);
}
```

The source code looks like the following.

```
//Created by MyEclipse Struts
//XSL source (default):
platform:/plugin/com.genuitec.eclipse.cross.easyst.struts.eclipse_3.8.4/xslt/JavaClass.xs
1

package de.laliluna.library.struts.form;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import de.laliluna.library.Book;
import de.laliluna.library.Customer;

/**
 * MyEclipse Struts Creation date: 07-27-2005
 *
 * XDoclet definition:
 *
 * @struts:form name="bookEditForm"
 */
public class BookEditForm extends ActionForm {

    // ----- Instance
    // Variables
```

```

private Integer customerId;

private Book book;

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

public Boolean getBorrowallowed() {
    return book.getBorrowallowed();
}

public Customer getCustomer() {
    return book.getCustomer();
}

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

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

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

public void setBorrowallowed(Boolean borrowallowed) {
    book.setBorrowallowed(borrowallowed);
}

public void setCustomer(Customer customer) {
    book.setCustomer(customer);
}

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

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

// ----- Methods
public Integer getCustomerId() {
    return customerId;
}

public void setCustomerId(Integer customerId) {
    this.customerId = customerId;
}

public Book getBook() {
    return book;
}

public void setBook(Book book) {
    this.book = book;
}

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

```

9.1.4 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"));
        // [laliluna] 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));
        // [laliluna] 28.11.2004 save customers in the request
        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"));

        // [laliluna] 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 logic
        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;

        // [laliluna] 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.

9.2 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:

New

Struts 1.1 Form Declaration
Create Struts 1.1 FormBean

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

Use case: customerList

Name: customerListForm

Form Impl: New FormBean Existing FormBean Dynamic FormBean

Superclass: org.apache.struts.action.ActionForm

Form type: de.laliluna.library.struts.form.CustomerListForm

Optional Details

Form Properties **Methods** JSP

Create methods?

- public ActionErrors validate(HttpServletRequest...)
- public void reset(HttpServletRequest...)
- public ActionErrors validate(ServletRequest...)
- public void reset(ServletRequest...)

< Back Next > Finish Cancel

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

Optional Details

Form Properties **Methods** **JSP**

Create JSP form?

New JSP Path: /jsp/customerList.jsp

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

New

Struts Action Declaration

Create Struts 1.1 Action

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

Use case: customerList

Path: /customerList

Action Type: Type Forward Include

Action Impl: Create new Action class Use existing Action class

Superclass: org.apache.struts.action.Action

Type: de.laliluna.library.struts.action.CustomerListAction

Optional Details

Form | Parameter | Methods | Forwards | Exceptions

Name: customerListForm Browse...

Attribute: customerListForm

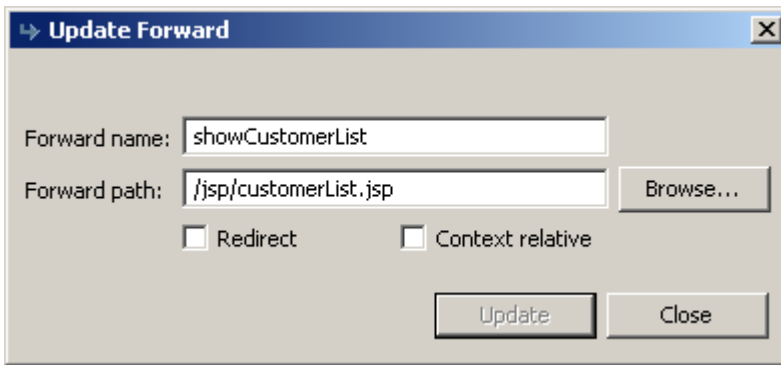
Scope: request

Validate Form

Input Source: Browse...

< Back Next > Finish Cancel

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" />
```

9.2.1 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;
        // [laliluna] 29.11.2004 get business logic
        LibraryManager libraryManager = new LibraryManager();

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

9.2.2 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>
          </tr>
        </logic:present>
      </tbody>
    </table>
  </body>
</html>
```

```

        </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="firstname" />
            </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=prepareEdit"
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 iterate -->
</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=prepareAdd'">Add a new customer
    </html:button>
    &nbsp;
    <html:button property="back" onclick="location.href='default.do'">Back to menu
    </html:button>
</body>
</html>

```

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

9.3 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.

The screenshot shows the 'New Form, Action and JSP' wizard. The 'Config/Module' field is set to `/LibraryWeb/WebRoot/WEB-INF/struts-config.xml`. The 'Use case' field contains `customrEdit`. The 'Path' field is `/customerEdit`. Under 'Action Type', the 'Type' radio button is selected. Under 'Action Impl', the 'Create new Action class' radio button is selected. The 'Superclass' dropdown is set to `org.apache.struts.actions.DispatchAction`. The 'Type' field is `de.laliluna.library.action.CustomerEditAction`. The 'Optional Details' section has tabs for 'Form', 'Parameter', 'Methods', 'Forwards', and 'Exceptions'. The 'Form' tab is active, showing 'Name' as `customrEditForm`, 'Attribute' as `customerEditForm`, 'Scope' as `request`, and 'Input Source' as `/jsp/customerEdit.jsp`. There is an unchecked checkbox for 'Validate Form'.

Then select to create a parameter:

The screenshot shows the 'Optional Details' section with the 'Parameter' tab selected. The 'Parameter' field contains the text `do`.

Then create two forwards as shown below.

The screenshot shows the 'Optional Details' section with the 'Forwards' tab selected. The 'Forwards' field contains two entries:

- customerList - [./customerList.do] redirect
- editCustomer - [./jsp/customerEdit.jsp]

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" />
```

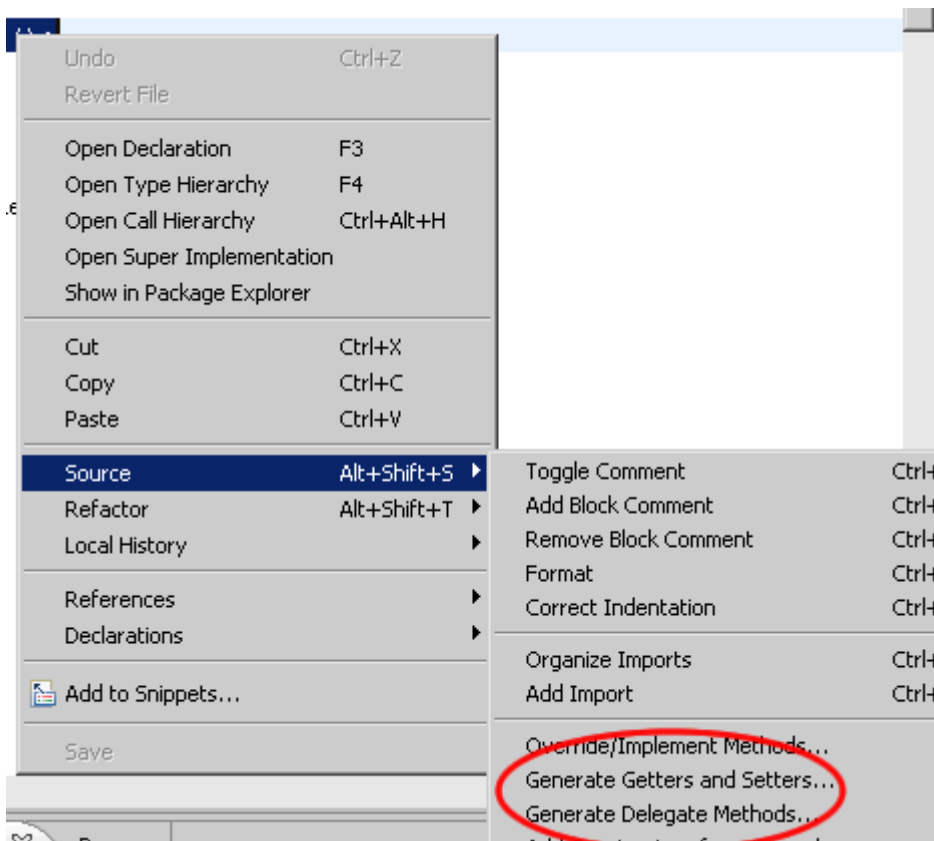
```
<action
  attribute="customerEditForm"
  input="/jsp/customerEdit.jsp"
  name="customerEditForm"
  parameter="do"
  path="/customerEdit"
  scope="request"
  type="de.laliluna.library.struts.action.CustomerEditAction"
  validate="false">
  <forward
    name="customerList"
    path="/customerList.do"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

9.3.1 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

```
package de.laliluna.library.struts.form;

import java.util.List;
import java.util.Set;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import de.laliluna.library.Customer;

public class CustomerEditForm extends ActionForm {

    // ----- Instance Variables
    private Customer customer;
    // ----- Methods

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }

    public boolean equals(Object rhs) {
        return customer.equals(rhs);
    }

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

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

    public String getLastname() {
        return customer.getLastname();
    }

    public String getFirstname() {
        return customer.getFirstname();
    }

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

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

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

    public void setLastname(String lastname) {
        customer.setLastname(lastname);
    }

    public void setFirstname(String firstName) {
        customer.setFirstname(firstName);
    }
}
```

```

public Set getBooks() {
    return customer.getBooks();
}

public void setBooks(Set books) {
    customer.setBooks(books);
}

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

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

    customer = new Customer();
}
}

```

9.3.2 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.

9.3.3 Edit the source code of the jsp file

Create a new file named *customerEdit.jsp* in the folder *WebRoot/jsp/*.

Open the file *customerEdit.jsp* and change the content of the file.

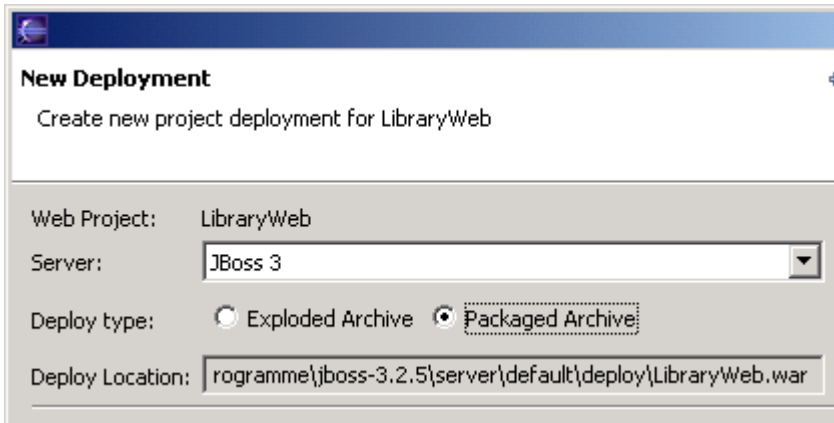
```
<%@ page language="java"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>

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

```
<html:submit/><html:cancel/>
</html:form>
</body>
</html>
```

10 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.

11 Copyright and disclaimer

This tutorial is copyright of Sebastian Hennebrueder, laliluna.de. You may download a tutorial for your own personal use but not redistribute it. You must not remove or modify this copyright notice. The tutorial is provided as is. I do not give any warranty or guaranty any fitness for a particular purpose. In no event shall I be liable to any party for direct, indirect, special, incidental, or consequential damages, including lost profits, arising out of the use of this tutorial, even if I has been advised of the possibility of such damage.