

Work in Progress

```
import java.sql.*;
```

```
public class CreateCoffees {  
    public static void main(String args[]) {  
        String url = "jdbc:mySubprotocol:myDataSource";  
        Connection con;  
        String createString;  
        createString = "create table COFFEES "  
            "(COF_NAME VARCHAR(40) "  
            "SUP_ID INTEGER, "  
            "PRICE FLOAT, "  
            "SALES INTEGER, "  
            "TOTAL INTEGER)";
```

```
        Statement stmt;
```

```
        try {  
            Class.forName("myDriver");  
        } catch (java.lang.ClassNotFoundException e) {  
            System.err.print("ClassNotFoundException: ");  
            System.err.println(e.getMessage());  
        }
```

```
        con.close();
```

```
    } catch (SQLException ex) {  
        System.err.println("SQLException: " + ex.getMessage());  
    }  
}
```

Struts Survival Guide

J2EE Survival Series

Basics to Best Practices

Companion Workbook

Srikanth Shenoy

Struts Survival Guide

***Basics to Best Practices
Companion Workbook***

Srikanth Shenoy



ObjectSource
Austin

ObjectSource LLC books are available for bulk purchases for corporations and other organizations. The publisher offers discounts when ordered in bulk. For more information please contact:

Sales Department
ObjectSource LLC.
2811 La Frontera Blvd., Suite 517
Austin, TX 78728

Email: sales@objectsource.com

First edition copyright ©2004 ObjectSource LLC. All rights reserved.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and ObjectSource LLC, was aware of a trademark claim, the designations have been printed in initial capital letters.

The author and the publisher have taken care in preparation of this book, but make no express or implied warranty of any kind and assume no responsibility for errors or omissions. In no event shall the ObjectSource LLC or the authors be liable for any direct, indirect, incidental, special, exemplary or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of use of the information or programs contained herein.

Published by
ObjectSource LLC
2811 La Frontera Blvd., Suite 517,
Austin TX 78728

Library of Congress Catalog Number: 2004100026

ISBN: 0-9748488-0-8 (paperback)

Printed in the United States of America

Table of Contents

Companion Workbook	1
Introduction	9
Exercise 1	11
Technical Objectives for this exercise:	11
Exercise 2	19
Business Objectives:	19
Technical Objectives for this exercise:	19
Exercise 3	26
Business Objectives:	26
Technical Objectives for this exercise:	26
Exercise 4	30
Technical Objectives for this exercise:	30
Exercise 5	31
Business Objectives:	31
Technical Objectives for this exercise:	31
Exercise 6	37
Business Objectives:	37
Technical Objectives for this exercise:	37
Exercise 7	39
Business Objectives:	39
Technical Objectives for this exercise:	39
Exercise 8	43
Technical Objectives for this exercise:	43
Exercise 9	45
Technical Objectives for this exercise:	45
Exercise 10	46
Technical Objectives for this exercise:	46

Preface

I started using Struts in late 2000. I was immediately drawn to its power and ease of use. In early 2001, I landed in a multi-year J2EE project, a large project by any measures. Struts 1.0 was chosen as the framework for the web tier in that project. Recently that project upgraded to Struts 1.1. I did the upgrade over a day. It cannot get any easier!

The thin book (let) you are holding is a companion workbook for the Struts Survival Guide, the Struts book that I wrote nearly six months ago. That book was pretty compact and yet covered everything that is there to Struts. Going by the reader's reviews I think I did a pretty good job ☺. Yet I thought I should write a companion workbook that would make readers better understand what I am talking in that book. In addition, I had to provide some Struts training to one of my clients. The workbook is a result of these two catalysts. This workbook can be used independently too, but makes a lot of sense when used in conjunction with the actual Struts Survival Guide. In the true spirit of open source, I am giving away this book for free.

I have enjoyed a lot writing this workbook as much as I enjoyed writing the original book. If you are a beginner, I bet my Struts Survival Guide is your fastest track to master Struts. Combined with this workbook, you have a killer combination. **This is still a work in progress.** But I decided to release it so that I can update it as I make progress and yet allow the readers to benefit from this.

I owe thanks to my wife for being so understanding when I had to work on this workbook during evenings and weekends. Finally I owe a lot to God through whom all things are made possible.

Srikanth Shenoy
July 2004

If you like the workbook, please support us by buying the book “Struts Survival Guide – Basics to Best Practices**”. It costs just \$14.95. It is available at Amazon.com, Barnes & Nobles and objectsource.com web sites. We are committed to publishing great books @ great prices. If you want Struts and J2EE training in your area, contact training@objectsource.com.**

How to use this workbook

The workbook contains 10 exercises in total. The very first exercise is about building your first Struts application. The steps for Exercise1 are not covered in the workbook since they are covered in detail in the actual book.

Each exercise has a set of technical objectives. Each exercise builds upon the previous exercise. Each exercise also contains detailed steps to achieve those objectives from the previous exercise. For instance, you can take Exercise 1 and apply the steps listed under Exercise 2 to achieve the technical objectives for Exercise 2.

Unzip the struts-training.zip to your C:\ drive. It creates a folder named “struts-training” with all exercises underneath it. The exercises were designed to work off the shelf with Eclipse. You can import the project into eclipse workspace and select the option “Run Ant” by right clicking on build.xml for each exercise to build and deploy the exercise directly to your favorite app server. The exercises were tested on WebLogic 8.1 SP2 running on Windows 2000. Please follow the environment setup section for detailed instructions. [You can also use it in non-eclipse environments. Since eclipse already compiles the java files, I skipped that step in the build.xml. You can add javac task to the build.xml and use it in non-eclipse environments]

Introduction

Environment Setup instructions

- 1) Check if Eclipse is already installed on your machine. (Some machines have Eclipse 2.1.1 installed. The training exercises were tested with 2.1.2.). Hence a reinstall is recommended. If you have Eclipse 2.1.1 already, then start by deleting the C:\eclipse directory.
 - 2) Install WebLogic 8.1 SP2
 - a. The exe is in [\\corpfile2\groups\Everyone\Training\Struts\binaries](#) folder
 - b. Launch the Installer and select Custom installation
 - c. Select only weblogic server (and examples) to be installed in the custom installation components screen
 - d. At the end, the installer asks if Node manager is needed. Say No
 - e. At the end of the installation uncheck the box for XMLSpy
 - f. Keep the quick start selected
 - 3) Create a WebLogic domain
 - a. The quick start will launch a domain creation wizard
 - b. Select options to create new domain New WebLogic Configuration --> Basic WebLogic Server Domain --> Express
 - c. Provide the username/password as weblogic/weblogic
 - d. Select Sun JDK to use
 - 4) Install Eclipse
 - a. The zip is in [\\corpfile2\groups\Everyone\Training\Struts\binaries](#) folder
 - b. Ensure that there is folder named "eclipse" in C:\
 - c. Extract the zip directly into C:\
 - d. Check if C:\eclipse directory is created
 - e. Launch eclipse.exe for initial workspace creation
 - f. Change the JDK used in Eclipse. Go to Window --> Java --> Installed JREs. Add the new JRE by setting the JRE Home as C:\bea812\jdk141_05
 - g. Also don't forget to check the new JRE checkbox
 - h. Close Eclipse
 - 5) Install Solar Eclipse (Used for editing XML and JSP)
 - a. Extract net.sf.solareclipse_0.4.1.bin.dist.zip into C:\eclipse
 - b. Reopen eclipse, and "finish" the update.
 - 6) Extract struts-training.zip directly into C:\. This will create a struts-training directory under C:\
-

7) Define a ODBC DSN

- a. Start --> Settings --> Control Panel --> Administrative Tools --> Data Sources (ODBC)
- b. System DSN --> Add --> Microsoft Access Driver (*.mdb) --> Finish
- c. DataSource Name: STRUTS_TRAINING
- d. Database Select: --> C:\struts-training\customers.mdb

8) Creating a Eclipse Project

- a. Copy C:\struts-training\exercise-archives\exercise01 folder into C:\struts-training
- b. In Eclipse, Select File --> Import – Existing Project into Workspace
- c. Select the C:\struts-training\exercise01 directory. If you have done this right, the project name should appear as exercise01
- d. Eclipse will complain about unbound classpath variable WEBLOGIC_LIB. You will have to define it as shown in the next step.
- e. Go to Window --> Preferences --> Classpath variables
- f. Select New. Set the name WEBLOGIC_LIB and path as: C:/bea812/weblogic81/server/lib/weblogic.jar
- g. Use the package explorer instead of the navigator. Select Window --> Open Perspective --> Java. This opens the Java perspective

9) Test the setup.

- a. Start weblogic by running startweblogic.cmd. It is present in c:\bea812\user_projects\domains\mydomain
 - b. Open C:\struts-training\environment.properties & correct the properties to match your local settings.
 - c. After weblogic is up & running, right click on build.xml and Select "Run Ant"
 - d. Check if the exercise01 is deployed. (There will be a "NoClassDef" error. Neglect it. If there are other errors, they need to be looked into).
 - e. Open the browser and go to <http://localhost:7001/exercise01>. If the application launches correctly, it implies JDK, weblogic and eclipse setup have worked. The DSN setup will be tested later in exercise04.
-

Exercise 1

Building your first Struts Application

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

1. Create Struts ActionForm and Action
 2. Understand the interaction between RequestProcessor, Action and ActionForm
 3. Start development with CustomerForm.jsp & add tags and TLDs [Using basic Struts tags - html and bean tags (totally 11 tags)]
 4. Understand the basic elements of struts-config.xml
 5. Create ActionForm fields for the JSP form fields
 6. Implement the validate method in ActionForm
 7. Create ActionMapping & associate the correct ActionForm and Action
 8. Create the Action class & implement the execute method
 9. Add ActionForward to the ActionMapping
 10. Understand how to use Message Resource Bundle
 11. Understand how to handle Form display, Form submission and validation
 12. Look for constants in JSP (bean:message, srcKey, altKey etc.) to add them in Resource Bundle
 13. Understand ForwardAction
 14. Look for ActionError keys & add them to Resource Bundle
 15. Build and deploy to WebLogic using the Build script provided
-

Let's start with developing your first Struts application. Here are the steps involved in creating the Struts application.

1. Add relevant entries into the web.xml
 - a. Add ActionServlet Configuration with initialization parameters
 - b. Add ActionServlet Mapping
 - c. Add relevant taglib declaration
2. Start with a blank template for the struts-config.xml. In the struts-config.xml, add the following
 - a. Declare the RequestProcessor
 - b. Create a properties file and declare it as Message Resource Bundle
 - c. Declare the Message Resource Bundle
 - d. Declare the Form-bean
 - e. Declare the ActionMapping for the Form-bean
 - f. Add the forwards in the ActionMapping
3. Create the Form-bean class
4. Create the JSP with Struts tags
5. Create the Action class
6. For every `<bean:message>` tag in the JSP, add key value pairs to the Message Resource Bundle (properties file) created in Step 3b
7. Add Validation in the Form-bean
8. Define the error messages in the Message Resource Bundle
9. Create the rest of the JSPs.

Next, you will find the steps to build the Struts application. You will find more explanation & rationale for the steps in the book Struts Survival Guide.

1. Add relevant entries into the web.xml

web.xml for the Struts Application

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>Hello World Struts Application</display-name>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
</web-app>
```

2) Create the struts-config.xml

struts-config.xml for the Struts Application

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
  <form-beans>
    <form-bean name="CustomerForm" type="struts.example.CustomerForm"/>
  </form-beans>

  <global-forwards>
    <forward name="mainpage" path="index.jsp" />
  </global-forwards>

  <action-mappings>
    <action path="/submitCustomerForm"
            type="struts.example.CustomerAction"
            name="CustomerForm"
            scope="request"
            validate="true"
            input="CustomerForm.jsp">
      <forward name="success" path="Success.jsp" />
    </action>
  </action-mappings>

  <controller processorClass="org.apache.struts.action.RequestProcessor"/>

  <message-resources parameter="struts.example.MessageResources"/>
</struts-config>
```

3) Create the ActionForm

CustomerForm

```
public class CustomerForm extends ActionForm {
  private String firstName;
  private String lastName;

  public CustomerForm() {
    firstName = "";
    lastName = "";
  }

  public String getFirstName() {
    return firstName;
  }
}
```

```
public void setFirstName(String s) {
    this.firstName = s;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String s) {
    this.lastName = s;
}
}
```

4) Create the CustomerForm JSP using Struts Tags

CustomerForm.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<html:html xhtml="true">
  <head>
    <title><bean:message key="exercise01.formpage.title"/></title>
    <html:base/>
  </head>

  <body background="images/blueAndWhiteBackground.gif">
    <h2><bean:message key="exercise01.formpage.title"/></h2>
    <html:errors/>
    <html:form action="/submitCustomerForm">
      <bean:message key="prompt.customer.firstname"/>:
      <html:text property="firstName" size="16" maxlength="16"/>
      <BR>
      <bean:message key="prompt.customer.lastname"/>:
      <html:text property="lastName" size="16" maxlength="16"/>
      <BR>
      <html:submit property="step">
        <bean:message key="button.save"/>
      </html:submit>
      &nbsp;
      <html:cancel>
        <bean:message key="button.cancel"/>
      </html:cancel>
    </html:form>
  </body>
</html:html>
```

5) Create the Action class

CustomerAction class

```
public class CustomerAction extends Action
{

    public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception
    {
        ActionForward nextPage = null;
        if (isCancelled(request)) {
            System.out.println("Cancel Operation Performed");
            return mapping.findForward("mainpage");
        }

        CustomerForm custForm = (CustomerForm) form;

        if ("Save".equals(custForm.getStep()))
        {
            String firstName = custForm.getFirstName();
            String lastName = custForm.getLastName();

            System.out.println("Customer First name is " + firstName);
            System.out.println("Customer Last name is " + lastName);

            nextPage = mapping.findForward("success");
        }
        return nextPage;
    }
}
```

6) Add properties to MessageResources.properties

Message Resource Bundle

```
#####
# Exercise01 index page strings
#####
exercise01.indexpage.title=Welcome to Exercise01

#####
# Exercise01 CustomerForm strings
#####
exercise01.formpage.title=Please enter your details
prompt.customer.firstname=First Name
prompt.customer.lastname=Last Name
button.save=Save
button.cancel=Cancel
```


7) Add validation to the Form bean

validate() method for CustomerForm

```
public ActionErrors validate(ActionMapping mapping, HttpServletRequest
request) {
    ActionErrors errors = new ActionErrors();

    // Firstname cannot be empty
    if (firstName == null || firstName.trim().equals("")) {
        errors.add("firstName", new ActionError("error.cust.firstname.empty"));
    }

    // Lastname cannot be empty
    if (lastName == null || lastName.trim().equals("")) {
        errors.add("lastName", new ActionError("error.cust.lastname.empty"));
    }

    return errors;
}
```

8) Add ActionError keys to the Message Resources

ActionError keys to Message Resources

```
#####
# Common
#####
errors.header=<h3><font color="red">Validation Error</font></h3>You must
correct the following error(s) before proceeding:<ul>
errors.footer=</ul><hr>
errors.prefix=<li>
errors.suffix=</li>

#####
# Exercise01 CustomerForm ActionErrors
#####
error.cust.firstname.empty=First Name is Required
error.cust.lastname.empty=Last Name is Required
```

9) Create the rest of the JSPs – index.jsp and Success.jsp. Notice that index.jsp uses the regular html:link tag that just forwards to another JSP. The Success.jsp uses the MVC compliant action mapping as the link. Define entries in MessageResource.properties for each of the bean:message keys in the JSPs.

index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
```

```
<html:html xhtml="true">
<head>
  <title><bean:message key="exercise01.indexpage.title"/></title>
  <html:base/>
</head>
<body background="images/blueAndWhiteBackground.gif">
  <div align="center">
    <html:link page="/CustomerForm.jsp">Go to Customer Form</html:link>
  </div>
</body>
</html:html>
```

Notice the usage of bean:write tags in Success.jsp. They let you access certain beans in appropriate scope and write their properties to the Servlet/JSP OutputStream

Success.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>

<html:html xhtml="true">
<head>
  <title><bean:message key="exercise01.successpage.title"/></title>
  <html:base/>
</head>
<body background="images/blueAndWhiteBackground.gif">
  <h2><bean:message key="exercise01.successpage.title" />
    <bean:write name="CustomerForm" property="firstName" />
    <bean:write name="CustomerForm" property="lastName" />
  </h2>
  <h3><bean:message key="exercise01.successpage.message" /></h3>

  <html:img src="images/beerchug.gif"/>

  <html:link page="/showCustomerForm.do">Go Back</html:link>
</body>
</html:html>
```

Exercise 2

Improving your first Struts Application

Business Objectives:

The following business objectives will be met at the end of this exercise:

1. Do not expose JSP URLs directly
2. Allow change of images without changing JSPs
3. Allow global change of error messages for a given type of error.
4. Add fields to capture customer address in the form using relevant form elements
5. Use Images instead of grey buttons for submitting data

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

1. Use MVC compliant `html:link` Tag
2. Use `Img` Tags (`html:img`) instead of ``. Externalize its `src` and `alt` to Resource Bundle.
3. Reuse error messages by value replacement
4. Modify the `ActionForm` and `JSP Struts Tags` to handle nested objects
5. Modify the `ActionForm`, `Struts Tags` in `JSP` & `Action` for image button form submission (using `ImageButtonBean`)
6. Use `Checkbox` & `Radio` tags for individual items
7. Initialize the `ActionForm` to pre-populate the `html` form
8. Use `Select` & `Option` Tag
9. Use `Options` to display collections in the `Select` Tag
10. Use Multiple Message Resource Bundles

1. Use MVC compliant `html:link` tag

- a. Change the link in `index.jsp` to `<html:link page="/showCustomerForm.do"/>` (In this case, the `ActionMapping` for `/showCustomerForm` is already defined in `struts-config.xml`. In general, this is also a step involved in making the link tags MVC compliant)

2. Use `html:img` tags and externalizing the `src` and `alt`

- a. We will externalize the image related information for the beerchug gif in the `Success.jsp` page.
-

- b. Add the following text to `MessageResources.properties` file:

```
image.beerchug=images/beerchug.gif
image.beerchug.alttext=It's Beer time
```

- c. Change the `` tag in `Success.jsp` to `<html:img srcKey="image.beerchug" altKey="image.beerchug.alttext"/>`.

3. Reuse error messages by value replacement (also how to access Resource Bundles programmatically)

Why: Between messages “First Name is Required” and “Last Name is Required”, only the field name changes. We can reuse the message as {0} is required. {0} is replaced with the display name for the field at runtime.

- a. Add the following to the `MessageResources.properties`.

```
error.required={0} is Required
```

- b. Remove the following error messages from `MessageResources.properties`

```
error.cust.firstname.empty=First Name is Required
error.cust.lastname.empty=Last Name is Required
```

- c. Change the `validate()` method in `CustomerForm` to use the following type of logic for validating each of first name and last name.

```
MessageResources msgRes = (MessageResources)
                           request.getAttribute(Globals.MESSAGES_KEY);

if (firstName == null || firstName.trim().equals(""))
{
    String firstName = msgRes.getMessage("prompt.customer.firstname");
    String[] rplcmntValueArr = { firstName };
    ActionError err = new ActionError("error.required", rplcmntValueArr);
    errors.add("firstName", err);
}
```

- d. Add similar logic for `lastName` validation also.

NOTE: An easier way to achieve the same result is as follows:

```
if (firstName == null || firstName.trim().equals(""))
{
    String[] rplcmntValueArr = { "First Name" };
    ActionError err = new ActionError("error.required", rplcmntValueArr);
    errors.add("firstName", err);
}
```

```
}
```

But with this approach, the display value of the field i.e. “First Name” is hardcoded in the ActionForm code. That is not good. We have the MessageResource.properties to externalize the display names for the field anyway. We can reuse it by accessing the MessageResources programmatically.

4. Modify the ActionForm and JSP Struts Tags to handle nested objects

- a. Create a class called `Address` in `struts.example` package. This is our nested object. Mark it as `java.io.Serializable` (**Why**: Because, at a later point we will put this object along with its encompassing ActionForm into Session. All Objects that go into HttpSession should be Serializable).
- b. Add the fields `address1`, `address2`, `city`, `state` and `zip` to the `Address` class. All the fields are strings
- c. Add the `address` field to `CustomerForm`. Also add getters for the `address` in `CustomerForm`. (**Why** only getters: The Struts framework will invoke `form.getAddress() .setXXX()` methods)
- d. Also add an `init()` method in `CustomerForm` and put all initialization code in there. Don't forget to initialize the `address = new Address()` (**Why**: Otherwise Struts throws a NullPointerException when it tries to set `form.getAddress() .setXXX()` methods).
- e. Call the `init()` method from `CustomerForm` constructor.
- f. In the JSP, add the text tags with nested properties to the existing customer form as follows:

```
<html:text property="address.city" />
<html:text property=" address.state" />
<html:text property=" address.zip" />
```

- g. Also don't forget to externalize their display labels to the Resource Bundle.
- h. You can test at this point to see if the nested property is working by adding some print statements to `CustomerAction` and deploy and test on WebLogic.

5. Use ImageButtonBean for Image based Form Submissions

Whenever a form is submitted via `<input type="image" name="save" />`, two request parameters `save.x` and `save.y` – corresponding to x and y coordinates are submitted. We don't worry about the exact value of the coordinates since we are not using image maps. Instead the mere presence of

save.x and/or save.y indicates that the form was submitted via the save image button. We combine this concept with Struts support for nested properties:

- a. Add two fields named save and cancel to CustomerForm. Both fields are of type ImageButtonBean. Add getter methods for each.
- b. Remove the step field and its getters and setters. We will not use it anymore.
- c. Change the CustomerAction to use the following check in its execute() method, instead of the form.getStep().equals("Save"):

```
form.getSave().isSelected()
```

This method checks if the save button is used for form submission.

- d. Similarly the isCancelled() method will not work since the Struts supplied cancel button checks for pre defined request parameter. We now have to replace it with:

```
form.getCancel().isSelected()
```

- e. Also since we are not using the pre-defined Cancel button anymore, the validate() method in CustomerForm will run for Cancel too. Hence add the code to bypass validation for Form Cancellation.
- f. In the CustomerForm.jsp, add the following for the image buttons:

```
<html:form>
..
..
..
<html:image property="save"
             srcKey="image.save"
             altKey="image.save.alttext" />

<html:image property="cancel"
             srcKey="image.cancel"
             altKey="image.cancel.alttext" />
..
</html:form>
```

- g. Define the srcKey and altKey for both images in the Resource Bundles

```
image.save=images/save.gif
image.save.alttext=Save the Form

image.cancel=images/cancel.gif
image.cancel.alttext=Cancel Submission
```

6. & 7. Use Checkbox & Radio Tags & Initialize/Prepopulate the Form

```
<html:options collection="STRUTS_EXAMPLE_STATES"
              property="value" labelProperty="label" />
</html:select>
```

NOTE: The tag is now looking for a collection as a page/request/session/application scope with the name **STRUTS_EXAMPLE_STATES**. Various options exist to create the collection and put it in one of the scopes.

- Create in the JSP page

- Create in the ActionForm and put it in request

- Create in the ActionForm and put in Session

- Create somewhere else and put in Application scope

The final option is the best since everybody uses the same read only data. We use a `ServletContextListener` to create the states in the application scope.

c. Copy the `ApplicationScopeInit` from the `exercise02` archive.

NOTE: This class creates a collection of `LabelValueBean`. It has two attributes: `value` and `label`. They are used in the `property` and `labelProperty` for options.

d. Change the `web.xml` to add the `ApplicationScopeInit` as a `ServletContextListener`. At the top, add:

```
<web-app>

  <display-name>Hello World Struts Application</display-name>

  <listener>
    <listener-class>
      struts.example.ApplicationScopeInit
    </listener-class>
  </listener>

  <servlet>
    ..
  </web-app>
```

e. This is a big step. Build, deploy and test to see if it works

10. Using Multiple Message Resource Bundles

NOTE: We will use a different message resource bundle to store information about images. For now, we will externalize the image related information for the `beerchug` gif in the `Success.jsp` page.

a. Create a new file called `ImageResources.properties` under `src/java/struts.example` directory (You can do this within Eclipse). Move the following entries from `MessageResources.properties` to `ImageResources.properties`:

```
image.beerchug=images/beerchug.gif  
image.beerchug.alttext=It's Beer time  
  
image.save=images/save.gif  
image.save.alttext=Save the Form  
  
image.cancel=images/cancel.gif  
image.cancel.alttext=Cancel Submission
```

- b. In the struts-config.xml, declare the new ImageResources.properties file as a Resource Bundle as follows:

```
<message-resources parameter="struts.example.ImageResources"  
                  key="bundle.image" null="false"/>
```

- c. For each of the image tag that corresponds to these images, change the tags in JSPs to use the **bundle="bundle.image"** Tags affected are: <html:img> in Success.jsp and 2 <html:image> tags in CustomerForm.jsp.

Deploy to WebLogic and Test

Get ready for Exercise 3

Exercise 3

Using JSTL, Struts-EL etc.

Business Objectives:

The following business objectives will be met at the end of this exercise:

1. Add support for JavaScript
2. Controlling field navigation without mouse
3. Greater Control over the look and feel using CSS Stylesheets
4. Additional Message Display in Success page if user has accepted to receive promotional email.
5. Simplifying JSP maintenance by eliminating buggy scriptlets.

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

11. Implement Focus on the first field
12. Implement javascript handlers such as onmouseover etc.
13. Use tabindex
14. Use CSS StyleClass, Style etc.
15. Use JSTL Tag `<c:out>`
16. Use JSTL Tag `<c:if>`
17. Use Struts-EL
18. Use `<c:forEach>` tag & implement an effective way to display Collection of Radio Boxes using `<c:forEach>`

1. Implement focus on first field

- a. Change the `<html:form>` link to

```
<html:form action="/submitCustomerForm" focus="firstName">
```

2. Implement JavaScript handlers

- a. Add a JavaScript to CustomerForm.jsp as follows:
-

```

<script>
function showAlert()
{
    alert("Are you sure you dont want to receive email alerts");
}
</script>

```

- b. Add the onclick event handler to the receiveEmail checkbox as follows:

```

<html:checkbox property="receiveEmail" onclick="showAlert()" />

```

3. Use tabIndex

- a. Add a tabIndex="x" (where x is an integer > 0) to any and all fields that you want to control

4. Use CSS Stylesheet

- a. Add `styleClass="mytxtbox"` to the email address field. The style mytxtbox is defined in the CSS that is bundled and deployed to WebLogic

Build, Deploy & Test at this point for sanity check

5. Use c:out

- a. Replace all `<bean:write>` with `<c:out>`. bean:write is of the format:

```

<bean:write name="CustomerForm" property="firstName" />

```

Replace it with:

```

<c:out value='${CustomerForm.firstName}' />

```

Look for all occurrences of `<bean:write>` & replace it with JSTL equivalents.

- b. Since this is the first time JSTL is being used, the TLD has to be defined in web.xml and also in each of the JSPs needing the tag.

```

<taglib>
  <taglib-uri>/WEB-INF/c.tld</taglib-uri>
  <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>

```

- c. Defined the TLD in each of the JSPs needing the tag. (CustomerForm.jsp, Success.jsp):

```

<%@ taglib uri="/WEB-INF/c.tld" prefix="c" %>

```

6. Use c:if (This will also illustrate a problem with checkbox)

NOTE: In this step, we will display a message to the customer if he has accepted to receive promotional emails. The CustomerForm has the checkbox for the customer to decide on receiving emails. We will use this information in the following page (Success.jsp) to control the message display.

- a. Add the following code to Success.jsp:

```
<c:if test='${CustomerForm.receiveEmail == true}' >
    <bean:message key="successpage.thanks"/>
    <br/>
    <bean:message key="successpage.emaildest"/>
    <c:out value='${CustomerForm.emailAddress}' />
</c:if>
```

The above tag uses EL to check if the request attribute CustomerForm receiveEmail is true and then print the additional message using <c:out>

- b. Build, Deploy & Test the application. You will find that no matter whether you check/uncheck the checkbox the message is always shown.

- c. Fix this by adding the following code to CustomerForm:

```
public void reset(ActionMapping mapping, HttpServletRequest request)
{
    if (request.getParameter("receiveEmail") == null ||
        request.getParameter("receiveEmail").equals("false"))
    {
        //Checkbox parameter was not received. Set the
        //receiveEmail to false
        receiveEmail = false;
    }
}
```

Why? That's due to how HTTP protocol treats checkboxes. When a checkbox is unchecked, that request parameter is not submitted. Since the CustomerForm is always created by setting the checkbox = true and it never arrives in the request again, the receiveEmail never becomes false. The above code is a way of making that happen.

Note: The above logic will not work for multipage form

- d. to the email `styleClass="mytxtbox"` to the email address field. The style mytxtbox is defined in the CSS that is bundled and deployed to WebLogic
-

7. Use Struts-EL

- a. Add the Struts-EL TLDs to web.xml:

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean-el.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean-el.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/WEB-INF/struts-html-el.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-html-el.tld</taglib-location>
</taglib>

<taglib>
  <taglib-uri>/WEB-INF/struts-logic-el.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-logic-el.tld</taglib-location>
</taglib>
```

- b. We will use only html-el tag library. Hence add the following declaration to Success.jsp and CustomerForm.jsp

```
<%@ taglib uri="/WEB-INF/struts-html-el.tld" prefix="html-el" %>
```

- c. Change some html tags to html-el in Success.jsp

8. Use c:forEach Tag

- a. Instead of hardcoding the enumeration for Carriers (FedEx, USPS etc.) for html:radio, we can use the <c:forEach> tag to iterate over a collection to display the radio boxes as follows:

```
<c:forEach var='carrier' items='${STRUTS_EXAMPLE_CARRIERS}'>
  <html-el:radio property="preferredCarrier"
    value="${carrier.value}"/>
  <c:out value='${carrier.label}' /> &nbsp;
</c:forEach>
```

- b. The above collection **STRUTS_EXAMPLE_CARRIERS** has also been created in ApplicationScopeInit class. Copy the modified class from exercise 3 archives.

NOTE: If you notice, the above collection has been named **STRUTS_EXAMPLE_CARRIERS** and not **STRUTS.EXAMPLE.CARRIERS**. The reason is that the collection name is used within EL. And the dot (.) has a special meaning within EL. Keep this in mind when you use EL.

Deploy to WebLogic and Test

Get ready for Exercise 4

Exercise 4

Applying Gof and J2EE Patterns

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

1. A whole lot of patterns based code is added in this exercise. Hence no Struts related functionality is added. Just the Search Functionality with Database is Implemented
2. Gof Patterns - Factory Method, Abstract Factory are implemented
3. Core J2EE Patterns - Business Delegate, Session Facade, Transfer Object, Data Access Object are implemented

Observe how patterns are applied and the Search functionality is implemented. This steps are not listed since they are not core to Struts usage that we want to illustrate. Time permitting, I will add this section later. Let us move on to important coverage of Struts features in subsequent exercises.

Deploy to WebLogic and Test

Get ready for Exercise 5

Exercise 5

Search, List, Action Chaining, Editable List Form

Business Objectives:

The following business objectives will be met at the end of this exercise:

1. From the search results, click thru for customer details
2. Provide features to add, edit and delete customers
3. Add hours of operation for each day of week for Customer

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

1. Reuse Actions by passing the context via Request Parameters
2. Pre populate Form from database.
3. Regular Action Chaining (Clearly understand when to use and when not to)
4. Action Chaining by parameter passing
5. Use Multibox for a collection of checkbox
6. Use Editable List Form

NOTE: Toughest and most elaborate exercise of all. Illustrates some very important concepts in Struts

1. Reuse Actions by passing context via Request parameters (Also covers pre populating form from database)

- a. In the List page, provide hyperlink to the email address to go to /showCustomerForm.do as follows:

```
<html-el:link page="/showCustomerForm.do?action=Edit
                &amp;email=${customer.emailAddress}">
    <c:out value='${customer.emailAddress}' />
</html-el:link>
```

- b. Currently /showCustomerForm.do is mapped to a ForwardAction. Now we need more than a ForwardAction to handle it. Create a full-fledged action called ShowCustomerAction in struts.example package as follows:

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
                             HttpServletRequest request, HttpServletResponse
                             response)
    throws Exception
{
    ActionForward nextPage = null;

    if ("Create".equals(request.getParameter("action")))
    {
        nextPage = mapping.findForward("customerFormPage");
    }
    else if ("Edit".equals(request.getParameter("action")))
    {
        CustomerDelegate delegate = new CustomerDelegate();
        CustomerDetailObject customer =
            delegate.getCustomerDetail(request.getParameter("email"));
        CustomerForm custForm =
            CustomerFormAssembler.createCustomerForm(customer);
        request.setAttribute("CustomerForm", custForm);
        nextPage = mapping.findForward("customerFormPage");
    }

    return nextPage;
}
```

- c. In the above code, the request parameter “Edit” was passed to indicate the context that a existing Customer is being updated and a blank form is shown.
- d. Create struts-config Action Mapping for the above action as follows:

```
<action path="/showCustomerForm"
        type="struts.example.ShowCustomerAction"
        scope="request"
        validate="false">
    <forward name="customerFormPage" path="/CustomerForm.jsp" />
</action>
```

- e. Build, Deploy & Test to see if the Customer is updated.
-

NOTE:

1. Notice that there is no `ActionForm` in the above declaration. When there is no associated form, the Action gets a null object passed in. Also note that the above action decides on what to do depending on the context. The final destination page is same in both cases.
2. Also note that this also demonstrates the form pre-population from database when editing.

3. Use Multibox

- a. Multibox is a collection of checkbox. When a single checkbox exists, it should mandatorily mapped to a Boolean. Such condition does not apply to collection of checkboxes. Here we map it to a String array
- b. Recall that Checkbox cannot exist outside a form. Hence start by defining a form to handle the checkboxes around the search display list as follows (Simplified for clarity):

```
<html:form action="/manageCustomerList">
<c:if test='${not empty CUSTOMER_SUMMARY_OBJECTS}'>
..
    <html-el:multibox property="idSelections">
        <c:out value='${customer.id}'/>
    </html-el:multibox>
..
</c:if>
</html:form>
```

- c. Define `ManageCustomersForm` to hold the checkbox information and initialize the string array as follows.

```
public class ManageCustomersForm extends ActionForm {
    private String[] idSelections;
    //getters and setters

    public ManageCustomersForm() {
        init();
    }

    protected void init() {
        idSelections = new String[] { "" };
    }
..
}
```

- d. Create a `ManageCustomersAction` to handle it in the package `struts.example.search`. Leave its `execute()` method empty.

- e. Add an ActionMapping in struts-config.xml. to associate /manageCustomerList.do with ManageCustomersAction and ManageCustomersForm

Build Deploy and See how the page displays at this point. This should give you a clear picture of using multibox.

4. Action Chaining (Delete Functionlity)

Action Chaining is a powerful facility and when used correctly you can elegantly design the web application control and navigation in Struts.

- a. Add a Delete button to CustomerSerachList.jsp in the ManageCustomers Form. The delete button should appear only when there are some search results.

```
<c:if test='${not empty CUSTOMER_SUMMARY_OBJECTS}'>
    <html:image property="deleteButton" srcKey="image.delete"
        altKey="image.delete.alttext" bundle="bundle.image"/>
</c:if>
```

- b. Add delete button to ManageCustomersForm
- c. Modify execute() method in ManageCustomersAction. In the execute(), invoke the appropriate Business Delegate method to delete all the selected customers. When the form is submitted, the idSelections field in the form has all the ids to be deleted.
- d. After successful deletion, forward the request to the following ActionForward by adding it to struts-config xml

```
<forward name="deleteSuccess" path="/submitCustomerSearchForm.do" />
```

This simple concept of forwarding to an action from another is called Action Chaining. Here the search page with the search results has to be displayed again after deletion.

A brute force way to achieve this is to perform search in the ManageCustomersAction and then attach the results in the appropriate scope or by calling a utility method in CustomerSearchAction. A better way is Action Chaining

Build, Deploy and Test

5. Action Chaining (Save Customer Form Functionality)

- a. Another example of Action Chaining is with the Custom erAction. After saving a customer successfully, (either adding a new or editing existing one), the user should be shown the search
-

page again possibly with the Search repeated. The way to achieve this by chaining to /submitCustomerSearchForm from CustomerAction.

6. Action Invocation by parameter passing (Add/Edit Customer – Big Picture)

- a. In the List page, add button for New. The new button should always show up no matter what. (unlike delete button that appears only when any search result exist).

```
<html:image property="newButton" srcKey="image.new"
            altKey="image.new.alttext" bundle="bundle.image"/>
```

- b. Add the newButton property to ManageCustomersForm
- c. In ManageCustomersAction, add a if block to handle new button. In that block, add a forward to /showCustomerForm.do?action=Create (by adding the same as a ActionForward).
- d. Now compare the href /showCustomerForm?action=Edit directly invoked from Customer List and the /showCustomerForm?action=Create indirectly invoked from the ManageCustomersAction and the usefulness of Action Invocation by parameter passing becomes clear

7. Use Editable List Form(Using Indexed properties of the tags)

- a. Create a class called HourOfOperation with dayOfWeek, openingTime and closingTime attributes. (You have made it Serializable haven't you ☺)
- b. Add a java.util.List called hoursOfOperationList to CustomerForm. Add a getter method
- c. Also add a method as follows:

```
public HourOfOperation getTiming(int index) {
    return (HourOfOperation) hourOfOperationList.get(index);
}
```

- d. Initialize the list by adding 7 HourOfOperation objects corresponding to each day of week.
- e. In the JSP, add the following

```
<c:forEach var='timing' items='${CustomerForm.hourOfOperationList}'>
    <tr>
        <td><bean:write name="timing" property="dayName"/></td>
        <td><html:text name="timing" property="openingTime"
                        indexed="true"/></td>
        <td><html:text name="timing" property="closingTime"
                        indexed="true"/></td>
    </tr>
</c:forEach>
```

Each of the form element in the iteration generates a item as follows:

```
<input type="text" name="timing[i].openingTime" value="" />
<input type="text" name="timing[i].closingTime" value="" />
```

where $i = 0$ to 6

This in turn invokes `getTiming(i).setOpeningTime()` and `getTiming(i).setClosingTime()` method (see c above) with the index $i = 0$ to 6. This will set the value of the `HourOfOperations` for all items in the List

- f. We have not added a table to persist this data. To check the availability of this data on server side add some print statements to check
- g. Also try initializing one or more of the `HourOfOperation` in the `CustomerForm` constructor and see how the initialized value is displayed on the html form

Deploy to WebLogic and Test

Get ready for Exercise 6

Exercise 6

Paging

Business Objectives:

The following business objectives will be met at the end of this exercise:

1. Implement Paging

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn how to implement Paging using Pager taglib

- a. Start by adding pager taglib to web.xml and CustomerSearchList.jsp

```
<taglib>
    <taglib-uri>/WEB-INF/pager-taglib.tld</taglib-uri>
    <taglib-location>/WEB-INF/pager-taglib.tld</taglib-location>
</taglib>
```

```
<%@ taglib uri="/WEB-INF/pager-taglib.tld" prefix="pg" %>
```

- b. Use <pg:pager> body tag to cover the entire list display

```
<pg:pager url="customerlist.do" maxIndexPages="10" maxPageItems="5">
```

- c. Use <pg:item> for each item iterated over

```
<c:forEach var='customer' items='${CUSTOMER_SUMMARY_OBJECTS}'>
    <pg:item>
        ....
    </pg:item>
</c:forEach>
```

- d. Add a new table containing <pg:index> block with the tags: <pg:prev>, <pg:pages>, <pg:next>

```
<TABLE width="80%" border="0">
    <TR><TD>&nbsp;</TD></TR>
    <TR align="center">
        <TD>
            <pg:index>
                <pg:prev><a href="<%= pageUrl%>">[&lt;&lt;Prev]</a></pg:prev>
                <pg:pages>
                    <a href="<%= pageUrl %>"><%= pageNumber %></a>
                </pg:pages>
                <pg:next><a href="<%= pageUrl%>">[Next&gt;&gt;]</a></pg:next>
```

```
        </pg:index>
      </TD>
    </TR>
  </TABLE>
```

Exercise 7

Better Form and Action Handling

Business Objectives:

The following business objectives will be met at the end of this exercise:

1. Split customer form into multiple pages
2. Prevent the customer from going back and resubmitting and getting a database error.
3. Better Exception handling in the system

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

19. To fix the bug: when Enter key is pressed in the Customer Search Form, a blank page is shown
20. Use DispatchAction
21. See how things work without exception handling mechanism
22. Provide exception handling mechanism (DuplicateCustomerException)
23. Create custom exception handlers
24. Provide Global exception handling mechanism with JSP
25. Create base ActionForm and Action Classes and use them
26. Create multipage forms. (Expln Only)
27. Handle Duplicate Form Submissions for CustomerForm (Expln Only)
28. Use DynaActionForm for CustomerSearchForm (Expln Only)

Default enter does not do search in Customer Search Form

1. When Enter key is pressed in the Customer Search Form, a blank page is shown

- a. This happens because the CustomerSearchAction performs the search only if serch.x or search.y are not null. Remove the if condition and it will work for enter key

2. Using DispatchAction

- a. Create two methods Create and Edit (notice the case) in ShowCustomerAction so that each method taking exactly the same input parameters as the execute method.
- b. Add appropriate code into each of these methods
- c. Comment (or delete) the execute method

- d. In `struts-config.xml`, add `parameter="action"` to the `ActionMapping` corresponding to `/showCustomerForm`.

3. See how the system works without exception handling

- a. Build and deploy. Test the `DispatchAction`. In addition try to add a new customer with the same email address as an existing customer. The database has unique constraint on email address and should throw an exception. See how the raw exception shows up.

4. Provide formatted exception handling for `DuplicateCustomerException`

- a. Add a exception block to the `ActionMapping` for `/submitCustomerForm` as follows:

```
<exception key="error.database.customer.duplicate"
           type="struts.example.customer.dao.DuplicateCustomerException"
           path="/CustomerForm.jsp"/>
```

The above xml block says that when the `Duplicate CustomerForm Exception` occurs the `CustomerForm JSP` has to be shown to the user. The error message is picked up from the `MessageResources` using the supplied key.

- b. Build and deploy. You will see that the Exception shows up as an `ActionError` since the page contains `<html:errors/>` tag already. However notice that the email address is `NULL`. The Struts exception-handling mechanism cannot do value replacement. A solution for this is to create a custom exception handler and do the value replacement as needed.

5. Create Custom Exception Handlers

- a. Create a new class called `DuplicateCustomerExceptionHandler` by subclassing the Struts `ExceptionHandler` class. Override the `execute()` method as follows:

```
public class DplicateCustomerExceptionHandler extends ExceptionHandler {
{
    public void ActionForward execute(Exception ex, ExceptionConfig ae,
                                     ActionMapping mapping, ActionForm formInstance,
                                     HttpServletRequest request, HttpServletResponse response)
                                     throws ServletException
    {
        ActionForward forward = mapping.getInputForward();
        ActionErrors errors = new ActionErrors();

        DuplicateCustomerException dup = (DuplicateCustomerException) ex;

        ActionError error = new ActionError(dup.getUserMessageKey(),
```

```

        dup.getValueReplacementArray());
errors.add("emailAddress", error);

//Store the ActionsErrors in request
request.setAttribute(Globals.ERROR_KEY, errors);

//Store the Exception in request
request.setAttribute(Globals.EXCEPTION_KEY, ex);

return forward;
}
}

```

- b. If you have a good exception hierarchy in your system to handle application exceptions, checked system exceptions, unchecked system exceptions then the above code can be generalized
- c. Change the exception block added to the ActionMapping earlier

```

<exception key="error.database.customer.duplicate"
           type="struts.example.customer.dao.DuplicateCustomerException"
           handler="struts.example.DuplicateCustomerExceptionHandler"/>

```

6. Create a Global exception handling block

- a. Create a new global-exceptions block just underneath form beans definition block and add a global exception block to as follows:

```

<exception key="error.database.customer.save"
           type="struts.example.customer.CustomerServiceException"
           path="/DatabaseError.jsp" />

```

- b. The DatabaseError.jsp is not a error page from Servlet specification perspective. The exception is not a implicit variable in the page. It is attached as a custom request attribute by the Default Exception Handler. You will find that the exception handler does not work right. But it is used to demonstrate that system exceptions (checked and unchecked) can be centralized using a block like this and probably an exception handler

7. Create a Base Form and Base Action (No Need to do this)

- a. Create a BaseForm and BaseAction classes. The BaseForm doesn't do much. It is just a marker interface.
 - b. In the BaseAction, implement the execute() method to add a preprocess, process and postprocess method. Add logEntry and logExit methods to indicate that you entered and exited the BaseAction's execute method.
-

- c. Define preprocess, and postprocess as do nothing methods
- d. Declare process method as abstract so that the concrete subclasses can implement it

Exercise 8

Creating Struts Modules

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

Create Struts application modules and break monolithic struts module into pieces

NOTE:

1. For the sake of maintaining simplicity and clarity in rest of the exercise, we will not attempt to break the existing application into modules. Rather we will add a new module.
2. The index page belongs to the default module. From the index page, we will provide a link to go to module2. Module 2 will consist of a two dummy pages that can call each other and also have a link to return to default module.

a) Change the web.xml to define a new module named xyz as follows:

```
<init-param>  
  <param-name>config/xyz</param-name>  
  <param-value>/WEB-INF/struts-config-xyz.xml</param-value>  
</init-param>
```

b) Copy over the struts-config.xml to create a new struts-config-xyz.xml

c) Create a empty XYZ Message Resources

d) Clean up the struts-config-xyz.xml to retain only empty blocks for form-beans, action mappings etc. Add the message resource bundle definition to struts-config-xyz.xml.

e) Add SwitchAction to struts-config.xml & struts-config-xyz.xml as follows:

```
<action path="/switch" type="org.apache.struts.actions.SwitchAction"/>
```

f) Create a folder called xyz under the src/web directory. Create two jsps: xyz-page1.jsp & xyz-page2.jsp (Copy over index.jsp & change contents)

- g) Add bean:message for titles and define them in XYZMessageResources
 - h) In xyz-page1.jsp, add link to navigate to xyz-page2.jsp (Forward Action via html:link)
 - i) In xyz-page2.jsp, add link to navigate to xyz-page1.jsp (Forward Action via html:link)
 - j) In index.jsp, a link is added to go to xyz-page1.jsp in xyz module as follows:
<html:link forward="gotoXYZModule">Go to XYZ Module</html:link>
 - k) In xyz-page1.jsp, add a link to go to index.jsp in default module
<html:link forward="gotoDefaultModule">Go to Default Module</html:link>
 - l) Add global forward in struts-config.xml to go to xyz-page1.jsp
**<forward name="gotoXYZModule"
path="/switch.do?page=/page1.do&prefix=/xyz" />**
 - m) Add global forward in struts-config-xyz.xml to go to index.jsp
**<forward name="gotoDefaultModule"
path="/switch.do?page=/showCustomerSearchForm.do&prefix="/>**
-

Exercise 9

Using Commons Validator with Struts

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

- 1) Use Commons Validator to validate ActionForm
 - a) We will add validations to CustomerSearchForm
 - b) The first thing while using Commons Validator is to extend the ActionForm from a predefined ValidatorForm. Hence we defined

Exercise 10

Using Struts & Tiles

Technical Objectives for this exercise:

The technical objectives of this exercise is to learn “how to”:

1) Use Struts and Tiles

a) Add TilesRequestProcessor to struts-config.xml

```
<controller processorClass="org.apache.struts.tiles.TilesRequestProcessor" />
```

b) Define the Tiles Plugin

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
  <set-property property="moduleAware" value="true"/>
</plug-in>
```

c) Create an empty Tiles definition file tiles-defs.xml under /WEB-INF directory

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
<tiles-definitions>

</tiles-definitions>
```

d) Add the base definition to the tile definition xml

```
<definition name="base.definition" path="/Layout.jsp">
  <put name="title" value=""/>
  <put name="header" value="/common/header.jsp" />
  <put name="footer" value="/common/footer.jsp" />
  <put name="body" value="" />
</definition>
```

e) Create the definition for each of the three pages by extending the above definition and overriding the empty values

```
<definition name="search.page" extends="base.definition">
    <put name="title" value="Customer Search and List"/>
    <put name="body" value="/CustomerSearchList.jsp" />
</definition>

<definition name="customer.page" extends="base.definition">
    <put name="title" value="XYZ Co. Please enter your details"/>
    <put name="body" value="/CustomerForm.jsp" />
</definition>

<definition name="dberror.page" extends="base.definition">
    <put name="title" value="An error occurred"/>
    <put name="body" value="/DatabaseError.jsp" />
</definition>
```

f) Change the references to physical jsps in struts-config.xml and replace them with one of the above names. For instance, change all references to

“/CustomerForm.jsp” as “customer.page” [No “/” is required with Tiles.]

NOTE: There is no replacement for index.jsp. Let us keep it as is. Hence don't change references to index.jsp
