ObjectSource

Struts for J2EE Developers

Srikanth Shenoy

ObjectSource LLC.
(http://www.objectsource.com)

J2EE Consulting, Mentoring and Training

Audience

ObjectSource

- Developers familiar with web application development, but not familiar with Struts
- Developers with high level picture of Struts
- Developers with Struts experience (AdvanTGWeb developers)

Objectives

ObjectSource

At the end of this course

- You will know everything you need to develop **serious & industrial strength** Struts web applications by applying good design practices.

Course Organization

ObjectSource

Basics

- Overview of J2EE, Servlet & JSP specs
- The case for Struts & Overview of Struts
- Struts Lifecycle - detailed view
- Developing your first Struts web application using basic Struts tags

Course Organization (Contd.)

ObjectSource

Intermediate

- Differences between regular tags and body tags
- More Struts Tags (ImageTag, Bean, Logic, Html)
- JSTL, Expression Language(EL), Struts-EL
- Multi page Forms
- Applying J2EE patterns in Struts application design
- Form Validation using Commons Validator

Course Organization (Contd.)

ObjectSource

Advanced

- Exception Handling in Struts applications
- Page Traversal with Pager Taglib
- Editable List Forms
- Using Struts with Tiles
- DispatchAction
- I18N, LookupDispatchAction
- Multiple Modules, SwitchAction
- Customizing Struts
- Odds and Ends

ObjectSource

Hour 1: Introduction to Struts

Making the case: Why Struts is needed &
High level overview of Struts

Hollywood Principle

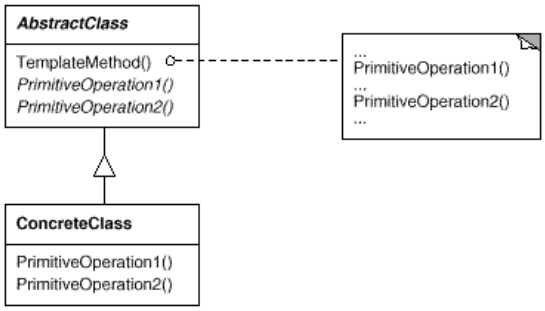
ObjectSource

- What is Hollywood Principle? (a.k.a. Inversion of Control)
- Don't Call us, We will call you
- Similarity to application development
- Of late, there is a lot of hype surrounding IoC

Template Method

ObjectSource

- A lot of us know it as Template Method Design Pattern



IoC – Its what J2EE is all about

ObjectSource

- Template Method a.k.a IoC is the basis for framework development
- Everything in J2EE is a Template Method – Servlets, EJBs all of them
- Everything in Struts is a Template Method

What is a Servlet?

ObjectSource

- A Servlet is a J2EE web component
- It is managed by a servlet container
- All Servlets implement the Servlet interface directly or indirectly.
- HttpServlet implements Servlet interface & handles HTTP requests
- Your servlets are sub-classes of HttpServlet class
- Your servlet class has to provide concrete implementations for doGet() and doPost() methods
- Differences between CGI & Servlets

What is a Servlet? (contd.)

ObjectSource

- A Servlet is mapped to one or more URL patterns. When the URLs match the pattern, the servlet container invokes doGet() or doPost() on that Servlet
- A sample doGet() method

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp) throws …. {
  OutputStream os = resp.getOutputStream();
  os.println("<HTML><BODY>Hello World</BODY><HTML>");
  os.flush();
  os.close();
}
```

Problems with Servlet?

ObjectSource

Problems with Servlet

- Presentation Logic & View are mixed up.
- Division on Labor cannot be achieved without Separation of concerns

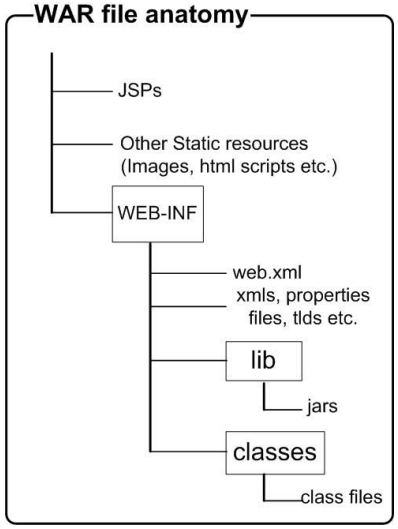
Solution: Use JSPs

What is a JSP?

ObjectSource

- JSPs are Servlets in disguise
- JSP separates Presentation Logic and View (How?)
- In addition JSP Tags eliminate scriptlet based logic in JSPs

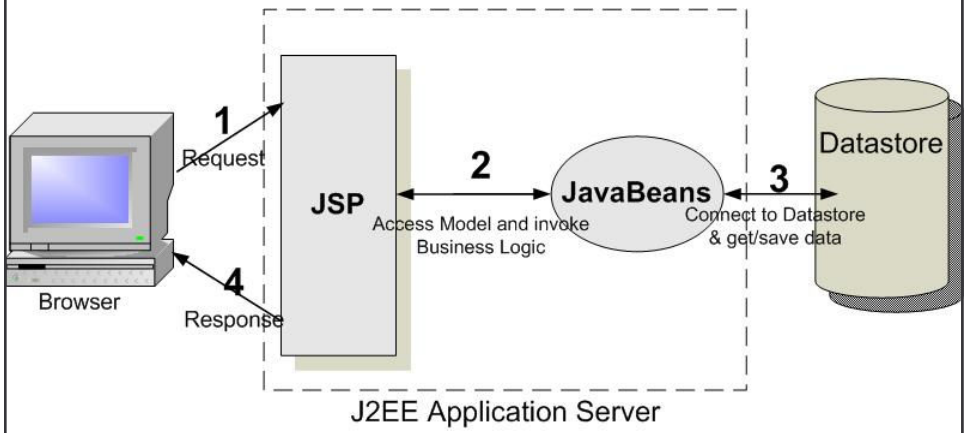
Anatomy of J2EE web application



JSP Programming models

- Model1 Architecture
- Model2 Architecture (MVC for Web)

Model 1 Architecture



Model 1 Architecture (contd.)

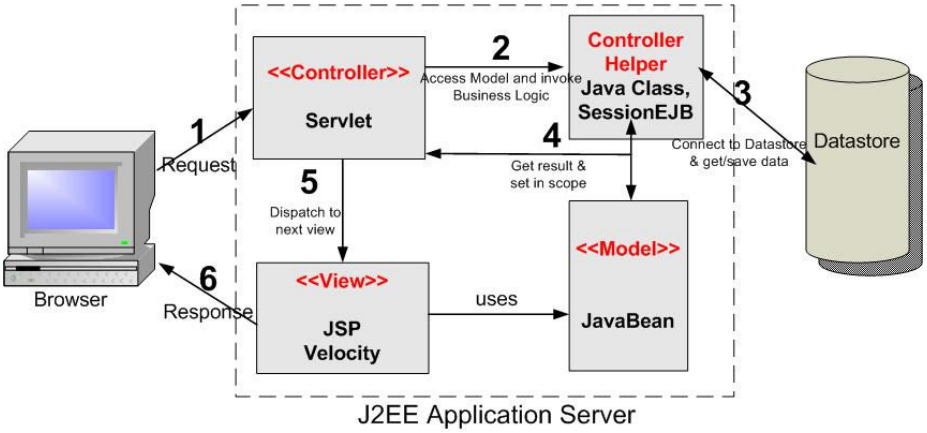
Pros

- Easy (Quick and dirty is the right description !!)

Cons

- Content, Presentation Logic & View Logic have some separation good enough only for small applications
- Separation of concerns is not fully achieved
- Navigation control is decentralized since every page contains the logic to determine the next page

Model 2 Architecture (MVC)



Model 2 Architecture (contd.)

Pros

- Content, Presentation and View Logic are well separated
- Good separation of concern
- JSP links point to a logical URL mapped to the Controller. Navigation control is centralized.

Best Practice

- Use on Controller Servlet per application instead of multiple servlets.

Problems with programmatic MVC

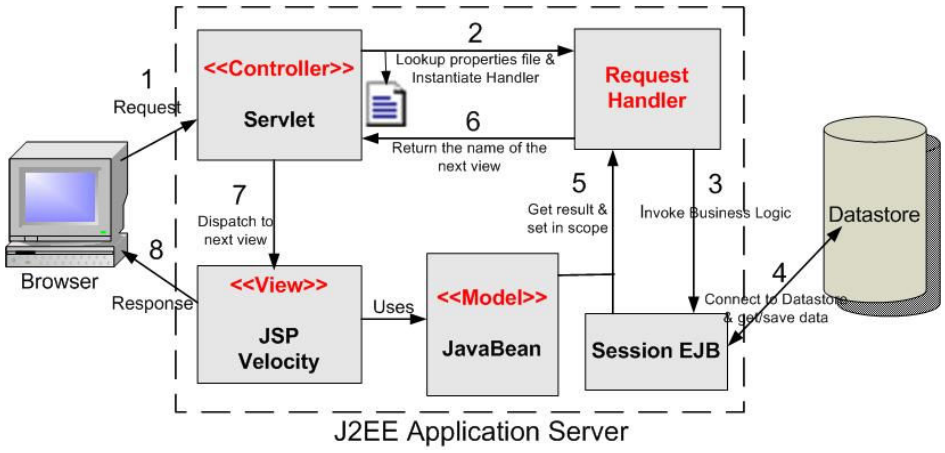
ObjectSource

- Lots of if blocks due to Stateless nature of HTTP and Request Response Model
- Results in **Fat Controller**
- High maintenance
- Changes to the Controller every time a new use case is added

So, What's the solution?

Enter Configurable MVC

ObjectSource



Configurable MVC Components

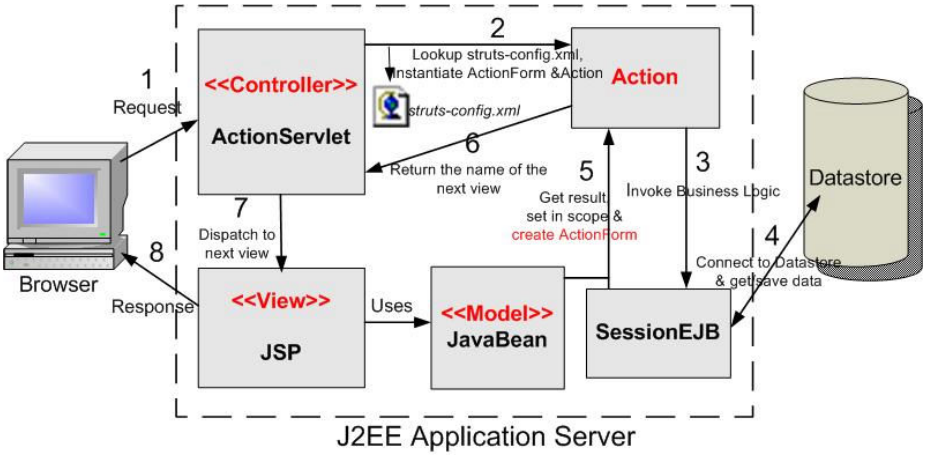
ObjectSource

- Declarative Approach
- Request Handlers
- Uses Hollywood Principle to work – “Don’t call us. We will call you”!!
- The rest of the stuff is pretty much same as regular MVC

So, what is Struts, after all

ObjectSource

Struts is nothing but one of many configurable MVC frameworks



What else does Struts provide

ObjectSource

- Controller is the main part of Struts but,
- Struts also provides Lots and Lots of tags useful in everyday JSP programming with Struts
- Struts controller can be used for any kind of view – Velocity, JSF etc.
- Provides Integration with Tiles
- Provides Integration with Commons Validator
- Provides custom extension points

Why Struts

ObjectSource

You can develop all that Struts provides by yourself,

BUT

Developing a framework

- *Feature-rich as Struts*
- *Testing & Debugging to make it robust*

is a massive project by itself.

Why reinvent the wheel when you get it for FREE ?

Remember...

ObjectSource

Struts is a product (& not a specification)

Remember not to tie your business logic into Struts. Keep it separate and decoupled from Struts and any javax.servlet.* classes

Hands-on: Installing Struts

ObjectSource

- Struts can be used with any J2EE compliant servlet container, We will use Struts with WebLogic
- Download Struts from http://jakarta.apache.org/struts
- Get familiarized with Struts directory structure
- “Self-Service” using Struts documentation (struts-documentation.war)
- Deploy Struts examples
- Use struts-blank.war as deployment template for your own code

Introduction to Struts

ObjectSource

Questions ??

ObjectSource

Struts Request Lifecycle

Detailed coverage of lifecycle + 1 exercise

(2 hours)

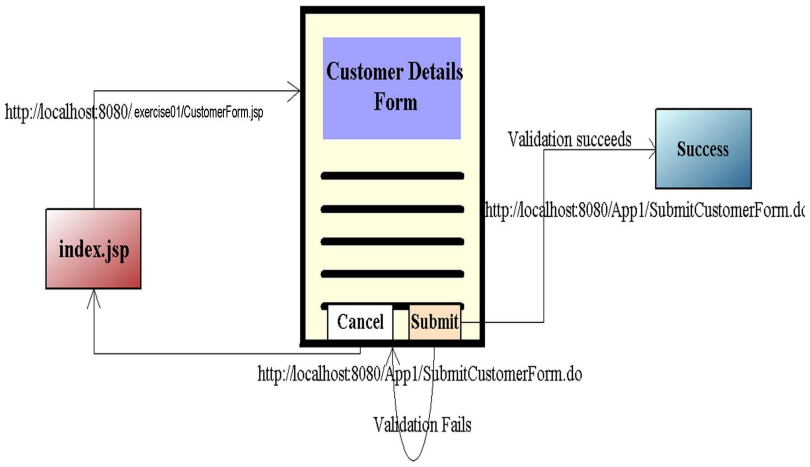
A working example

ObjectSource

- Explanation with example
- Open Eclipse. Import Exercise01
- Start WebLogic
- Run Ant in Eclipse to build & deploy exercise01.war to WebLogic

Page Navigation

ObjectSource



Example HTML Form

ObjectSource

```
<form  name="MyCustomerForm"
        action="/submitCustomerForm.do">

   <input type="text" name="firstName" value="" />
   <input type="text" name="lastName" value="" />

   <input type="submit" name="step" value="Save" />
</form>
```

Basics

ObjectSource

- Struts Request Lifecycle is based on Hollywood Principle
- Hollywood Principle – "Don't call us, We will call you"
- Good example of Template Method Design Pattern

Participating components

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- ActionForm
- Action
- ActionForward
- ActionErrors
- ActionMapping

struts-config.xml

ObjectSource

8 Important components

- *struts-config.xml – Struts Configuration file. One file per module*
- ActionServlet
- RequestProcessor
- ActionForm
- Action
- ActionForward
- ActionErrors
- ActionMapping

ActionServlet

ObjectSource

8 Important components

- struts-config.xml
- *ActionServlet – Only ONE servlet for every Struts web application (WAR)*
- RequestProcessor
- ActionForm
- Action
- ActionForward
- ActionErrors
- ActionMapping

ActionServlet (Contd.)

ObjectSource

- Only one for the entire application
- Reads struts-config.xml during initialization - `init()`
- Looks up the appropriate RequestProcessor in `doGet()` and `doPost()` methods
- Invokes RequestProcessor.`process()`

ActionServlet – web.xml setup

Servlet Definition

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-
  class>org.apache.struts.action.ActionServlet</s
  ervlet-class>
  <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-
  config.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

ActionServlet web.xml setup (contd.)

URL Pattern to Servlet Mapping

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

RequestProcessor

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- *RequestProcessor – Class that manages the request lifecycle. One RequestProcessor per module. (A WAR can have potentially multiple modules)*
- ActionForm
- Action
- ActionForward
- ActionErrors
- ActionMapping

ActionForm

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- *ActionForm – Object representation of the HTML Form. Attribute names should match the HTML Form field names*
- Action
- ActionForward
- ActionErrors
- ActionMapping

Action

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- ActionForm
- *Action – The Struts version of Request Handler. The RequestProcessor calls the execute() method. Only One Action instance of each type*
- ActionForward
- ActionErrors
- ActionMapping

ActionForward

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- ActionForm
- Action
- *ActionForward – Object representation of the next URL to forward to.*
- ActionErrors
- ActionMapping

ActionErrors

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- ActionForm
- Action
- ActionForward
- *ActionErrors – Object representation of errors in HTML Form*
- ActionMapping

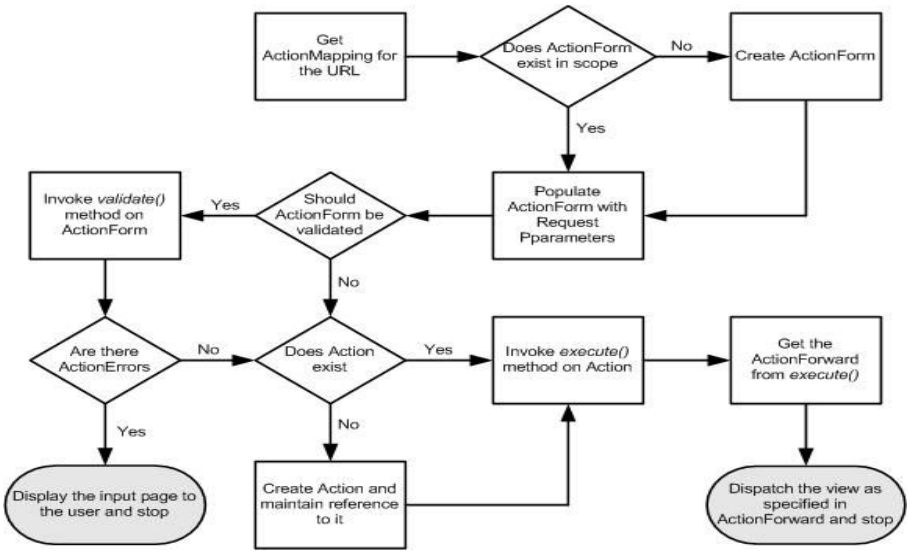
ActionMapping

ObjectSource

8 Important components

- struts-config.xml
- ActionServlet
- RequestProcessor
- ActionForm
- Action
- ActionForward
- ActionErrors
- *ActionMapping – Object that holds the information on which Action to invoke for a given URL and its associated ActionForm, whether to validate the Form etc.*

RequestProcessor process() flow



struts-config.xml

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

HTML Form & Struts interaction

ObjectSource

```
<form name="MyCustomerForm"
  action="/submitCustomerForm.do">

  <input type="text" name="firstName" value=""/>
  <input type="text" name="lastName" value=""/>

  <input type="submit" name="step" value="Save"/>
</form>
```

ActionForm

ObjectSource

```
public CustomerForm extends ActionForm {
  private String firstName, lastName, step;
  public ActionErrors validate(….) {
    ActionErrors errors = new ActionErrors();
    if (firstName == null ||
        firstName.trim().length() == 0) {
      ActionError err = new
      ActionError("firstName.required")
      errors.add("firstName",err);
    }
    return errors;
  }
}
```

Action

ObjectSource

```
public CustomerAction extends Action {
  public ActionForward execute(…..) {
      if (form.getStep().equals("Save")) {
        // Invoke the business logic here
        ActionForward f = new ActionForward();
        f.setName("Success");
        f.setPath("/Success.jsp");
        return f;
      }
  }
} // Two issues: Button Name hardcoded, JSP name hardcoded
```

Action reworked

ObjectSource

```
public CustomerAction extends Action {
  public ActionForward execute(…..) {
      if (form.getStep().equals("Save")) {
        // Invoke the business logic here
        ActionForward f =
              mapping.findForward("success");
        return f;
      }
  }
} // One issue remains: Button Name hardcoded
```

struts-config.xml – 8 sections

ObjectSource

1. Form Bean definition section
2. Global Forward definition section
3. Action Mapping definition section
4. Controller definition section
5. Application Resources definition section
6. Plugin definition section
7. DataSource definition section
8. Exception definition section

struts-config.xml sections

ObjectSource

```
<global-forwards>
    <forward   name="logoff"  path="/logoff.do"/>
</global-forwards>

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

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

So much for the controller...

ObjectSource

What about the view?

- How are null values displayed?
- How are errors displayed?
- How is the Form pre-populated from the Struts ActionForm?
- A particular case of pre-population: How are previously entered values redisplayed when there is an error in submitted form?

That's what Struts Tags are for

Struts Tag Categories (6)

ObjectSource

- HTML Tags
- Bean Tags
- Logic Tags
- Template Tags
- Nested Tags
- Tiles Tags

How Form Tag works

ObjectSource

```
<html:form action="/submitCustomerForm">
   <html:text property="firstName" />
   <html:text property="lastName" />
   <html:submit property="step">Save</html:submit>
</html:form>
```

becomes

```
<form name="MyCustomerForm"
   action="/submitCustomerForm.do">
   <input type="text" name="firstName" value=""/>
   <input type="text" name="lastName" value=""/>
   <input type="submit" name="step" value="Save"/>
</form>
```

Externalizing Strings

ObjectSource

- Hard coding field names & errors is a bad practice
- When externalized, they can be easily changed and reused.
- In Struts, externalized string are stored in properties file called Resource Bundle

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

- There can be multiple bundles per module with one default module
- `<bean:message>` tags is used to display resource bundle strings in JSP
- ActionError takes the key of the externalized String as the parameter

How Errors Tag works

ObjectSource

- `<html:errors/>` iterates over ActionErrors request attribute.
- It expects pre-defined properties in Resource Bundle
  - `errors.header=<h3>Errors</h3><ul>`
  - `errors.footer=</ul>`
  - `errors.prefix=<li>`
  - `errors.suffix=</li>`

Tags used in first exercise (11)

ObjectSource

- HTML HTMLTag - **<html:html>……</html:html>**
- HTML Base Tag - **<html:base/>**
- HTML LinkTag - **<html:link page=“”>**
- HTML ImageTag - **<html:img src=“”>**
- HTML FormTag - **<html:form action=“”>..</html:form>**
- HTML ErrorsTag - **<html:errors/>**
- HTML TextTag - **<html:text property=“”/>**
- HTML SubmitTag - **<html:submit property=“”/>**
- HTML CancelTag - **<html:cancel/>**
- Bean MessageTag - **<bean:message key=“”/>**
- Bean WriteTag - **<bean:write name=“” property=“”/>**

Hands-on: Developing your First Struts Web Application

- Use Eclipse as the IDE and look at the project named Exercise01
- Build using Ant. Ant script is directly under the top directory. Your ant path is also set. Just run Ant
- The application is automatically hot deployed by the ant build. By dropping the exercise01.war into WebLogic mydomain/applications

A closer look at Exercise 1

- Get a good picture of desired navigation
- Start development with CustomerForm.jsp & add tags and TLDs
- Create ActionForm for the JSP form fields
- Implement the validate method if needed
- Create ActionMapping for the Form & associate the correct ActionForm and Action
- Create the Action class & implement the execute method
- Go back and add ActionForward to the ActionMapping
- Look for constants in JSP (bean:message, srcKey, altKey etc.) to add them in Resource Bundle
- Look for ActionError constants & add them to Resource Bundle

Struts Request Lifecycle

ObjectSource

Questions ??

ObjectSource

Got Tags?

Detailed coverage of Struts Tags + 2 exercises

(2 hours)

Tag Lifecycle

ObjectSource

- Two types of Tags – Regular and Body Tags
- Servlet container creates pools for each Tag
- A tag is assigned to each invocation & attribute setter methods, doStartTag(), doEndTag() and release() are invoked in that order (in case of Regular Tags)
- Regular Tags can also have body but they do not interact with it
- BodyTag always has a body
- BodyTag passes context to the body
- BodyTag lifecycle is more complex and will not be covered in this course

Struts Tag Categories (6)

ObjectSource

- HTML Tags
- Bean Tags
- Logic Tags
- Template Tags
- Nested Tags
- Tiles Tags

How Form Tag works (Recap)

ObjectSource

```
<html:form action="/submitCustomerForm">
   <html:text property="firstName" />
   <html:text property="lastName" />
   <html:submit property="step">Save</html:submit>
</html:form>
```

becomes

```
<form name="MyCustomerForm"
   action="/submitCustomerForm.do">
   <input type="text" name="firstName" value=""/>
   <input type="text" name="lastName" value=""/>
   <input type="submit" name="step" value="Save"/>
</form>
```

How Form Tag works (Recap)

ObjectSource

- FormTag is a BodyTag
- FormTag passes the ActionForm context to the tags in its body namely TextTag, CheckboxTag etc.
- Tags such as TextTag etc. cannot exist independently

How Errors Tag works (Recap)

- `<html:errors/>` iterates over ActionErrors request attribute.
- ErrorsTag iterates over ActionErrors and displays them
- It expects pre-defined properties in Resource Bundle
  - `errors.header=<h3>Errors</h3><ul>`
  - `errors.footer=</ul>`
  - `errors.prefix=<li>`
  - `errors.suffix=</li>`

More on how Errors Tag works

- ActionErrors is the model for the ErrorsTag
- Each ActionError is associated with a form field or **GLOBAL_ERROR**
- Messages in Resource Bundle can be reused by value replacement technique.
- Messages are defined as {0} is required
- `new ActionError(String key, Object[] replacementValueArray)`
- Pass the replacement values in the replacement value array

Tags used in first exercise (11)

ObjectSource

- HTML HTMLTag - **<html:html>……</html:html>**
- HTML Base Tag - **<html:base/>**
- HTML LinkTag - **<html:link page=“”>**
- HTML ImageTag - **<html:img src=“”>**
- HTML FormTag - **<html:form action=“”>..</html:form>**
- HTML ErrorsTag - **<html:errors/>**
- HTML TextTag - **<html:text property=“”/>**
- HTML SubmitTag - **<html:submit property=“”/>**
- HTML CancelTag - **<html:cancel/>**
- Bean MessageTag - **<bean:message key=“”/>**
- Bean WriteTag - **<bean:write name=“” property=“”/>**

Don’t call JSP directly

ObjectSource

- Don’t call a JSP directly from another JSP
- It violates MVC (such as all links to the controller, centralized navigation etc.)
- In Struts it prevents the Controller from setting request attributes for Model and behavior control such as Locale, Resource Bundle, ActionErrors, Collections to iterate over etc.
- Use html:link tag to invoke the action mapping
- E.g. *<html:link page=“/showCustomerForm.do” />*
- Always start the URL with “/”

Why use Image Tag <html:img>

- To handle localized image and image alt text as follows:

```
<html:img srcKey="image.click"
          altKey="image.click.alttext" />
```

Becomes

```
<img src="click.gif" alt="Click here to get
your award"/>
```

- Also used for URL rewriting for role based or dynamic dynamic images

Nested property support

- Struts 1.1 provides nested property support for Tags & ActionForms
- Consider a Tag **<html:text property="address.city"/>**
- When a Form is displayed, the above tag displays the value: `getAddress().getCity()`
- When a Form is submitted, the above tag does
  - Retrieve the address object in the ActionForm
  - Invoke `getAddress().setCity()` on the address object
- No setter methods needed for Nested Objects in ActionForm
- Address has to be initialized in the ActionForm !!

Assignment

ObjectSource

Modify exercise01:

- Make use of MVC compliant **<html:link>** tag in index.jsp
- Externalize the image name and alt text for the **<html:img>** tag in Success.jsp
- Reuse the error messages for both Customer firstName and lastName using Value Replacement technique discussed earlier
- Add a "Address" nested object in CustomerForm with city, state and zip – all string fields.
- Use the text tags to collect the user data from Form

Assignment solution

ObjectSource

- *Change the link tags in index.jsp as follows*
  - <html:link page="/showCustomerForm.do" />
- *Externalize the image name and alt text for the <html:img> tag in Success.jsp*

a) Add the following to MessageResources
   - image.beerchug=images/beerchug.gif
   - image.beerchug.alttext=It's Beer time

b) Add the following to Success.jsp

```
<html:img srcKey="image.beerchug"
          altKey="image.beerchug.alttext" />
```

Assignment solution (contd.)

ObjectSource

- *Reuse the error messages for both Customer firstName and lastName using Value Replacement technique discussed earlier*

1) Add the following to MessagedResources

error.required={0} is Required

2) Remove the error messages for FirstName Required, Last Name Required from MessagedResources

3) On next page…..

Assignment solution (contd.)

ObjectSource

- *Reuse the error messages for both Customer firstName and lastName using Value Replacement technique discussed earlier*

3) Modify the validate method as follows:

```
MessageResources msgRes = (MessageResources)
   request.getAttribute(Globals.MESSAGES_KEY);
String firstName =
  msgRes.getMessage("prompt.customer.firstname");
String[] rplcmntValueArr = { firstName };
ActionError err = new
   ActionError("error.required",rplcmntValueArr);
```

Assignment solution (contd.)

ObjectSource

- *Add a "Address" nested object in CustomerForm with city, state and zip – all string fields.*

1) Create a Address class

```
public class Address implements Serializable {
    private String city, state, zip;
    getters and setters….
}
```

2) Modify the CustomerForm

```
public CustomerForm extends ActionForm {
    private Address address;
    getters and setters for address
}
```

Assignment solution (contd.)

ObjectSource

- *Use the text tags to collect the user data from Form*

Add the following text tags to the CustomerForm.jsp

```
<html:form action="/submitCustomerForm">
  <html:text property="firstName" />
  <html:text property="lastName" />
  <html:text property="address.city" />
  <html:text property=" address.state" />
  <html:text property=" address.zip" />
  <html:submit property="step">Save</html:submit>
</html:form>
```

Don't forget to externalize their labels!!!!

ImageButton for Form Submission

ObjectSource

```
<form name="MyCustomerForm"
        action="/submitCustomerForm.do">
   <input type="text" name="firstName" value="" />
   <input type="text" name="lastName" value="" />

   <input type="image" name="step"
                      src="images/save.gif" />
   <input type="image" name="cancel"
                      src="images/cancel.gif" />
</form>
```

What is submitted to the server?

ObjectSource

- When a image with name "**step**" is clicked, two request parameters – **step.x** and **step.y** are submitted

Struts way of ImageButton

ObjectSource

```
<html:form action="/submitCustomerForm">
   <html:text property="firstName" />
   <html:text property="lastName" />
   <html:image property="step"
               srcKey="images.save" />
</html:form>
```

- This Form submits two request parameters for the save image – step.x and step.y
- `step.x` & `step.y` – Sounds like nested properties!!
- Create any object with properties X and Y
- Use it as nested object in ActionForm

Impact of ImageButton on ActionForm

ObjectSource

```
public CustomerForm extends ActionForm {
   private String firstName, lastName;
   private ImageButtonBean step;

   public ImageButtonBean getStep() {
        return step;
   }
   //No need for setStep() method
}
```

ImageButtonBean

ObjectSource

```
public ImageButtonBean {
    private String x, y;
    public String getX() {..}
    public String setX() {..}
    public String getY() {..}
    public String setY() {..}

    public boolean isSelected() {
        return (x != null) || (y != null);
    }
}
```

Impact of ImageButton on Action

ObjectSource

```
public CustomerAction extends Action {
    public ActionForward execute(…..) {
        //Previously - if (form.getStep().equals("Save"))
        //Now
        if (form.getStep().isSelected() )
            mapping.findForward("success");
            return f;
        }
    }
}
```

Select & Option Tags

ObjectSource

```
<select name="state">
    <option value="AL">Alabama</option>
    <option value="AK">Alaska</option>
</select>
```

Can be done in brute force manner as

```
<html:select property="state">
    <html:option value="AL">Alabama</html:option>
    <html:option value="AK">Alaska</html:option>
</html:select>
```

OR……

Use Options Tag Instead

ObjectSource

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

- Create a collection of name value pairs
- Set the colelction in some scope with a name

Radio Button, Checkbox

ObjectSource

- Suppose that there is a option for the customer to select Gender. This is represented in HTML as: <input type="radio" name="gender" value="Male" checked/>
- Struts: <html:radio property="gender"/>
- The appropriate radio button is automatically selected based on the value of the ActionForm attribute
- For Checkbox, the ActionForm attribute has to be of type boolean

Attributes common to HTML Tags

ObjectSource

- property=".." except select & option tags
- name
- value (Used mostly with Radio box only)
- accessKey
- tabIndex
- alt, altKey
- style, styleClass
- JavaScript event handlers – onclick, onmouseover, onmouseout etc.

Struts Tag Categories (6)

ObjectSource

- HTML Tags
- ***Bean Tags***
- Logic Tags
- Template Tags
- Nested Tags
- Tiles Tags

Bean Tag Category

ObjectSource

- Multiple Message Resource Bundles & <bean:message>
- <bean:write>
- <bean:define>

- <bean:define>
- Format: <bean define id="x" name="beanName" property="propertyFromBean" />
- Exposes x as the scripting variable to be used for other tags and scriptlets

Multiple Resource Bundles

ObjectSource

- Multiple Resource Bundle declaration in struts-config

```
<message-resources parameter="struts.example.
  MessageResource" key="bundle.alt"/>
```

- Alternate resource bundle usage example

```
<bean:message key= "title.firstName" bundle=
  "bundle.alt" />
<html:errors bundle="bundle.errors" />
<html:img ... Bundle="bundle.images"/>
```

<bean:define>

ObjectSource

- Format:

```
<bean define id="x" name="beanName"
        property="propertyFromBean" />
```

- Exposes x as the scripting variable
- Exposed scripting variable can be used in scriptlets & other tags (implicitly or explicitly)

Another assignment

ObjectSource

Modify exercise01:

- Convert the state textbox into a drop down. Link it to a options collection
- Replace the grey buttons with Image Buttons in JSP
- Make appropriate code changes to CustomerForm and Action

Exercise 2

ObjectSource

Objectives

- Nested Properties
- Img Tag, Image Buttons
- Multiple Resource Bundles
- Radio, Select, Options and Checkbox Tags
- Reuse error messages by value replacement

Struts Tag Categories (6)

ObjectSource

- HTML Tags
- Bean Tags
- ***Logic Tags***
- Template Tags
- Nested Tags
- Tiles Tags

Logic Tags

ObjectSource

- <logic:iterate>
- <logic:equal>
- <logic:notEqual>
- <logic:greaterEqual>
- <logic:greaterThan>
- <logic:lessEqual>
- <logic:lessThan>

Logic Iterate Tag

ObjectSource

```
<logic:iterate id="customer" name="company"
   property="customers">
   // Execute for every element in the customers
   // collection in company bean.
   // Use the scripting variable named customer
   <bean:write name="customer"
                        property="firstName" />
</logic:iterate>
```

Common attributes of Logic Tags

ObjectSource

- name
- property
- value

- Example:

```
<logic:equal name="customer" property="firstName"
                                    value="John">
      //Do whatever needs to be done when
      Customer's first name is John
</logic:equal>
```

Limitations of Logic Tags

ObjectSource

- Gets very complicated for non-trivial logic checks
- Consider how you would represent this using tags:

```
if ( customer.firstName == "John" &&
      customer.lastName == "Doe" &&
      customer.age == 28)  {
    do something...
}
```

Limitations of Logic Tags (contd.)

ObjectSource

```
<logic:equal name="customer" property="firstName"
                                value="John">
  <logic:equal name="customer" property="lastName"
                                value="Doe">
    <logic:equal name="customer" property="age"
                                value="28">
      //do something….
    </logic:equal>
  </logic:equal>
</logic:equal>
```

Limitations of Logic Tags (contd.)

ObjectSource

- Logic Tags can only do Logical ANDing.
- There is no way to do Logical Oring
- That's where Expression Language (EL) comes in handy.

```
<c:if test='${customer.firstName == "John" &&
                customer.lastName == "Doe" &&
                customer.age == 28}'>
        //do something ...
</c:if>
```

Expression Language (EL)

ObjectSource

Here is how to do Logical OR

```
<c:if test='${ customer.firstName == "John" ||
                customer.lastName == "Doe" ||
                customer.age == 28}'>
        //do something ...
</c:if>
```

EL & JSTL

ObjectSource

- Expression Language is very much Java like
- Evaluates contents between ${ and }
- You cannot write long snippets of code as scriptlets (And that's good)
- EL is available as part of JSTL now. JSTL stands for JSP Standard Tag Library
- JSTL is a specification & not a product/framework

EL & JSTL (contd.)

ObjectSource

- JSTL 1.0 will work only in JSP 1.2 containers
- JSTL 1.1 will work only in JSP 2.0 containers
- EL will become part of JSP 2.0 specification & EL can be used in the entire JSP
- For now, EL can be used only within the JSTL tags
- We will use JSTL 1.0

JSTL Crash Course

ObjectSource

- JSTL has four categories
  - Core: Contains tags for if/then, output, iterating collections
  - Formatting: Contains Tags for I18N, setting Resource Bundles, formatting and parsing number, currency, date
  - SQL: Database Access Tags
  - XML: Tags for XML Parsing and Transforming with XPath
- We will only deal with Core

JSTL crash course (contd.)

ObjectSource

```
<% User user = (User)pageContext.findAttribute("user");
    if (user != null) {
      Role[] roles = user.getRoles();
%>
<ul> <% for (int i=0;i<roles.length;i++) { %>
       <li>Role Name is <%= roles[i].getName() %></li>
      <% }%>
</ul>
<% }%>
```

JSTL Crash Course (contd.)

ObjectSource

- Same JSP snippet with JSTL

```
<ul>
    <c:forEach items="${user.roles}" var="role">
        <li><c:out value=${role.name}/></li>
    </c:forEach>
</ul>
```

Expression Language

ObjectSource

- When JSTL encounters an EL variable
  - It first checks if the variable is a implicit object (In JSTL, implicit objects are predefined objects with a predefined name such as: pageContext, params etc.)
  - If not found, it checks for object with that name in Page, Request, Session scopes in that order (For e.g. user in the previous example is a object in the page scope)
  - The traverses the nested object hierarchy

Where to get JSTL Binaries?

ObjectSource

- JSTL standard interfaces and classes are defined in core.jar – Available from Sun
- Implementation for JSTL interfaces is vendor specific
- We will use an implementation from Jakarta
  - Uses core.jar from the JSTL specification
  - Uses standard.jar containing the expression engine from Jakarta TagLibs

Struts and JSTL together

ObjectSource

- You can use JSTL tags along with Struts tags
- But Struts has many tags without a JSTL equivalent
- That's where Struts-EL fits in
- Struts-EL is a port of Struts tags to use EL
- Available under /contrib in Struts 1.1 download
- Best of Both Worlds – Elegance of JSTL & Power of Struts

When to choose JSTL over Struts Tags?

ObjectSource

- Pure JSTL replacements
  - <c:forEach> instead of <logic:iterate>
  - <c:if> instead of <logic:equal>, <logic:notequal> etc.
  - <c:out> instead of <bean:write>
- Struts-EL replacements
  - <bean-el:message> instead of <bean:message>
  - <html-el:*> instead of <html:*> (Since JSTL has no equivalents for Struts HTML Tags)

Radio Box using Collection

ObjectSource

- Using LabelValueBean
- Displaying a collection of radio boxes

Exercise 3

ObjectSource

Objectives

- JSTL Tags - <c:if>, <c:out>
- Struts-EL – replace all regular struts tags with struts-el
- Using <c:forEach> for an effective way to display Collection of Radio Boxes
- Value Additions
  - Implementing Focus on the first field
  - Implementing javascript handlers such as onmouseover etc.
  - Using tabindex
  - Using CSS StyleClass, StyleId etc.

Got Tags?

ObjectSource

Questions ??

ObjectSource

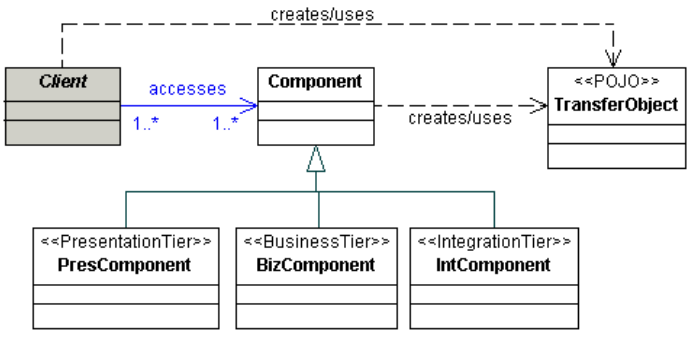
The Patterns Soup

Relevant Gof & J2EEPatterns + 1 exercise

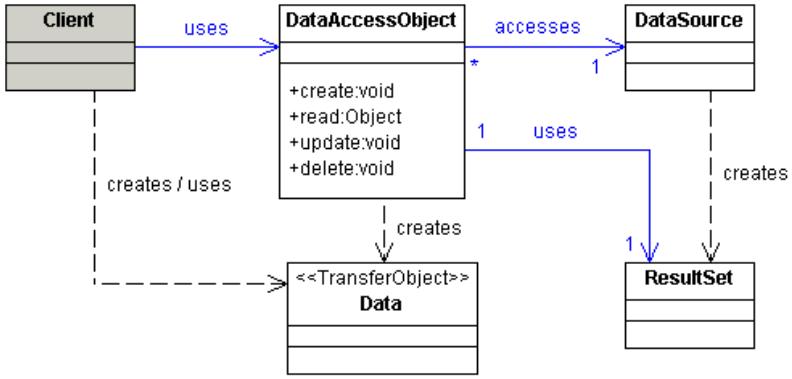
(1 hour)

Transfer Object

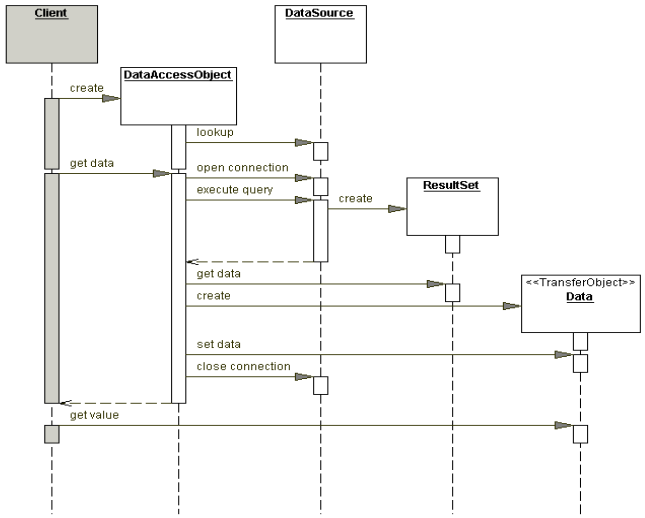
ObjectSource



Data Access Object



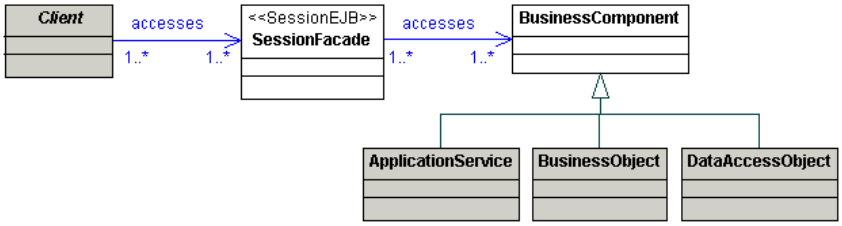
Data Access Object (contd.)



Session Façade

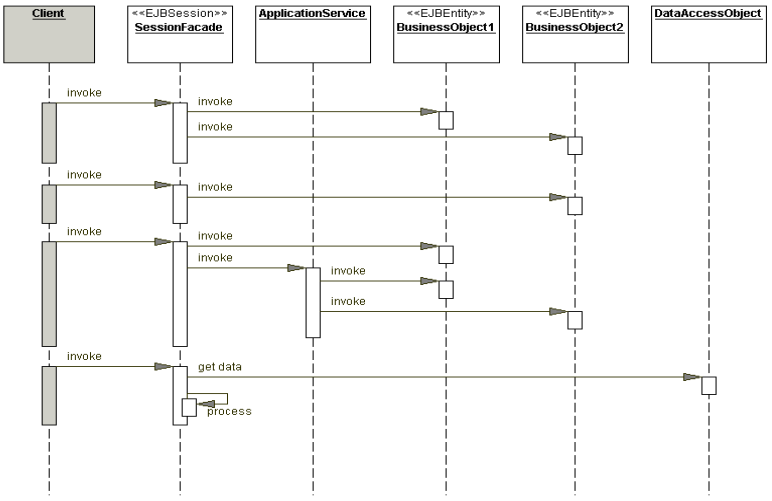
ObjectSource

- Implemented as Session EJB
- Reduces network roundtrips and centralizes remote access



Session Façade (contd.)

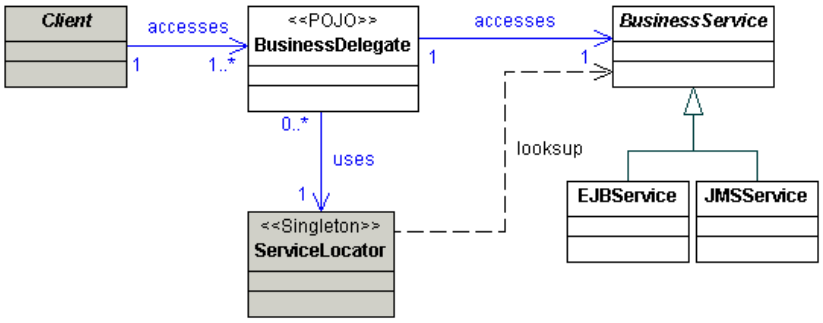
ObjectSource



Business Delegate

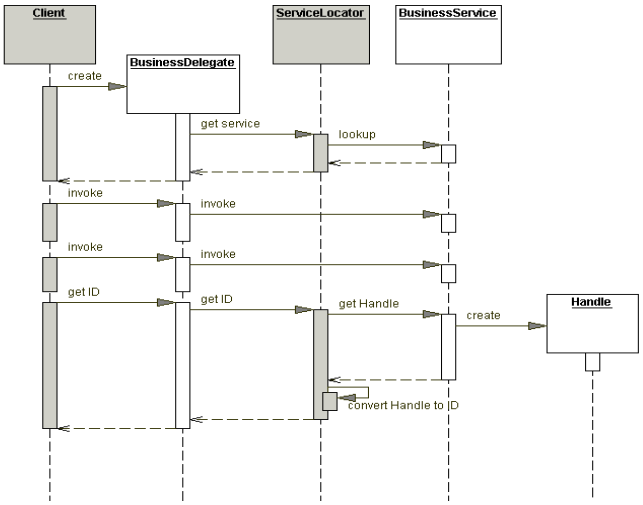
ObjectSource

- Making business logic access protocol independent

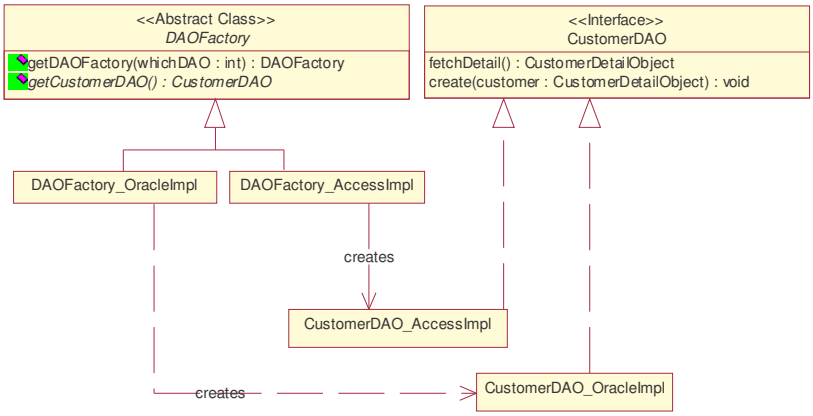


Business Delegate (contd.)

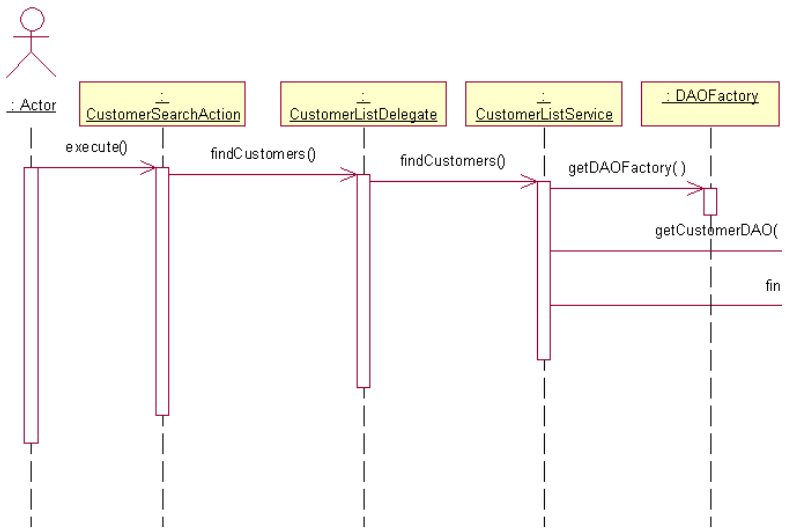
ObjectSource



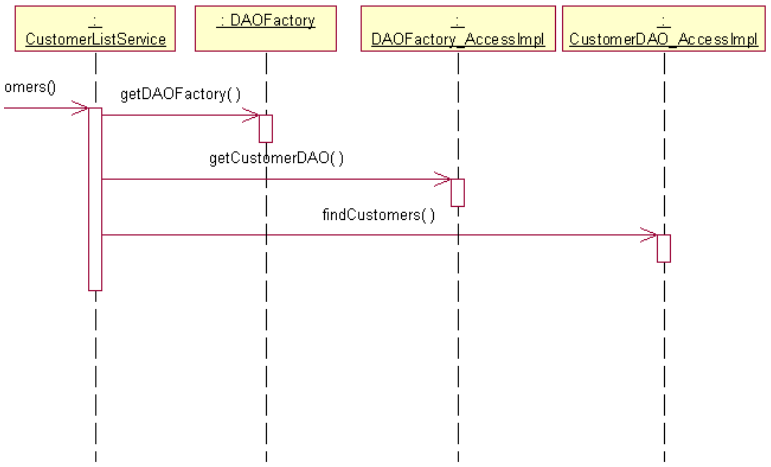
Gof pattern: Abstract Factory



Customer Search Sequence



Customer Search Sequence (contd.)



The Patterns Soup?

Questions ??

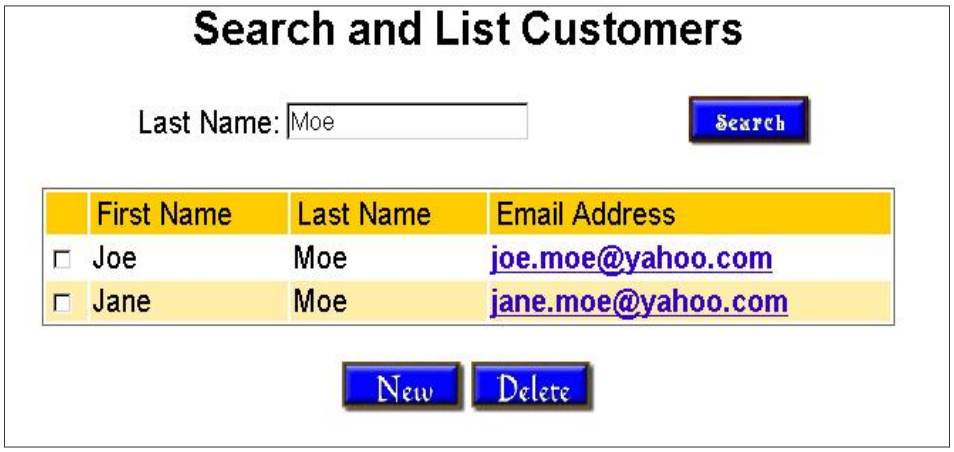
ObjectSource

Assorted Topics - Action Chaining, Multibox, Editable List Form, Paging

2 hours – Includes 2 exercises

Exercise 5 Demo

ObjectSource



Reusing Action

ObjectSource

- When CustomerForm is displayed for editing, it is pre-populated
- When CustomerForm is displayed for new customer creation, it is empty.
- Solution 1: Populate the FormBean in JSP.
- Solution 2: Reuse the Action by passing context
- Pass the context to ShowCustomerAction as
  - /showCustomerForm.do?action=Edit
  - /showCustomerForm.do?action=Create

Reusing Actions (contd.)

ObjectSource

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

- ShowCustomerAction will do
  - Form pre-population
  - Attach Form as request attribute &
  - Forward to the CustomerForm.jsp
- Notice: No ActionForm necessary, ActionForm is null

Collection of Checkboxes

ObjectSource

- A single checkbox is represented by <html:checkbox> and mapped to a boolean
- Collection of Checkboxes is represented by <html:multibox> and can map to anything – an array of Strings, a collection of integers
- Usage:

```
<html-el:multibox property="ids">
    <c:out value='${customer.id}'/>
</html-el:multibox>
```

- Creates <input type=checkbox name=ids value=2 /> and so on

Collection of Checkboxes (contd.)

ObjectSource

- Multiple HTTP request parameters with the same name “ids”
- Map the ids to a String array in the ActionForm

Action Chaining

ObjectSource

- Action Chaining: When one Action forwards to a Logical Page which is another Action by itself
- Usually done when Action B has to be always performed after Action A
- Examples:
  - A Search Action is done after every Delete Action
  - A Search Action is done after a Customer is saved
- Caution: Don't use Action Chaining as replacement for Session Facade

Indexed Properties

ObjectSource

- Example: <html:text property="phoneNum" indexed="true"/>
- Always used within a iterate or forEach tag.
- Creates the html as follows:

  <input type="text" name="phoneNum[i]" value="" />

  i = 0 to N
- Add the following indexed methods in ActionForm:

  public String getPhoneNum(int i) { … return ..}

  public void setPhoneNum(int i) { … return ..}

Indexed Properties (contd.)

ObjectSource

- Indexed Properties are not same as nested properties
- Nested Properties are mapped to nested objects
- Indexed Properties are mapped to collection or arrays of nested objects or primitives
- Multibox achieves “almost” the same thing for checkboxes. (Browsers don’t allow multiple elements with same names except for Checkboxes)
- Indexed properties can be used to handle editable list forms

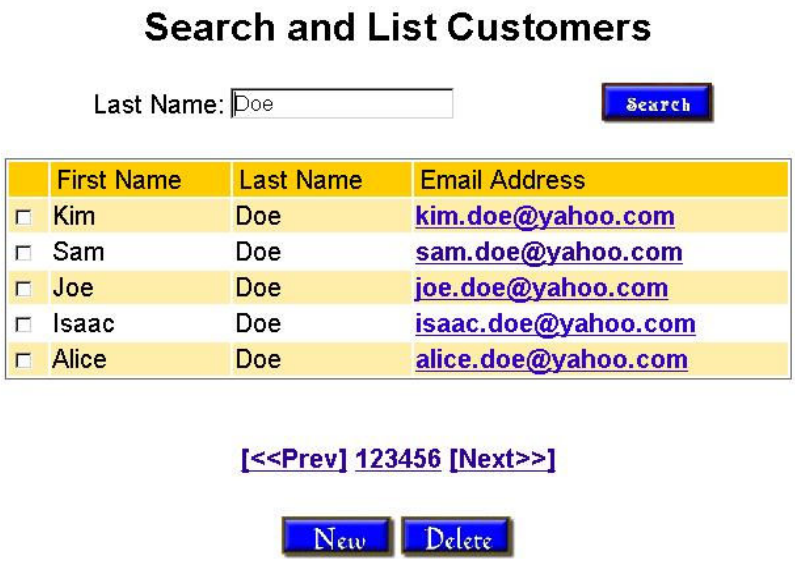
Time for Exercise 5

ObjectSource

- Reuse Action by parameter passing
- Action Chaining
- Multi box
- Editable List Forms using Indexed Properties

Paging

ObjectSource



Pager Taglib

ObjectSource

- We use Pager Taglib from jsptags.com for Paging
- Pager TagLib is LGPL
- Takes care of logic for page traversal.

Main Tags:

- Top level - <pg:pager>
- Individual Item display - <pg:item>
- Navigation Control
  - <pg:prev>
  - <pg:pages>
  - <pg:next>

Tags in Pager Taglib

ObjectSource

- pg:pager
  - Outermost tag.
  - Defines top level control entities such as Number of items per page etc
  - You also need to provide URL to be invoked on prev, next etc.
- pg:item - Controls number of items displayed
- pg:index
  - Controls the page navigation tags
  - Exposes several scripting variables such as pageUrl, page number to be used by pg:prev, pg:next and pg:pages
- pg:param – Lets you define any additional request parameters

Pager Taglib – Sample Code

ObjectSource

```
<pg:pager url="customerlist.do" maxIndexPages="10"
   maxPageItems="5">
  <c:forEach var='customer' items='${collection}'>
    <pg:item>
     //Display individual row data here
   </pg:item>
  </c:forEach>
  <pg:index>
   <pg:prev><%= pageUrl %></pg:prev>
   <pg:pages><%= pageUrl %><%= pageNumber%></pg:pages>
   <pg:next><%= pageUrl %></pg:next>
  </pg:index>
<pg:pager>
```

Paging

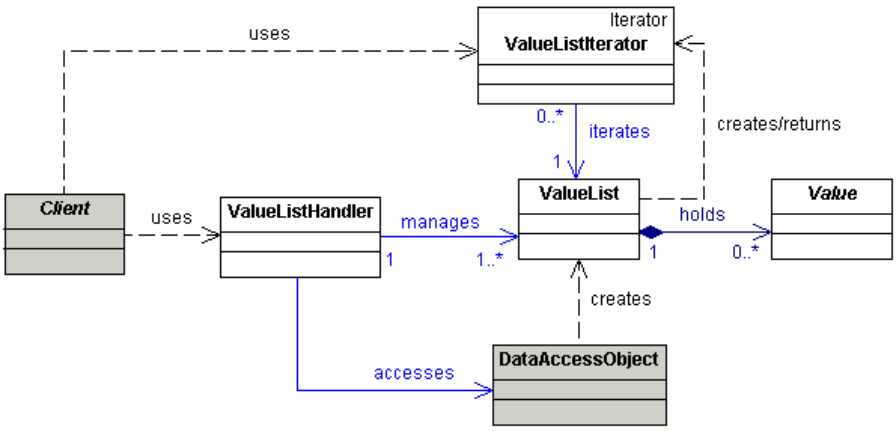
ObjectSource

- Exercise 6 stores the search results in HttpSession
- Lets Pager Taglib iterate over it
- Putting Search results in Session is never a good idea – Why?
- We will cover ValueListHandler pattern and a strategy for high performance page traversal

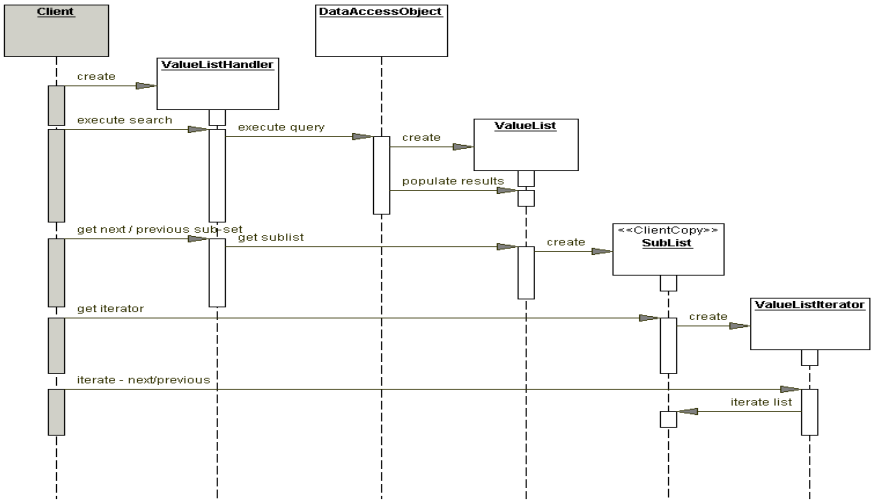
- ***Nevertheless, its time for Exercise 6***

ValueListHandler Pattern

ObjectSource



ValueListHandler (contd.)



High Performance Page Traversal

- Database Independent traversals – Almost impossible
- Database dependent
  - Slightly dependent – AdvabTGWeb DBList Framework
  - Fully dependent
- Luckily all databases support some form of tracking
  - DB2: ROW_NEXT
  - Oracle: ROW_NUM
  - Sybase: ROWCOUNT

High Performance Page Traversal for DB2

ObjectSource

- First Page

```
SELECT FIRST_NAME, LAST_NAME, ADDRESS
FROM CUSTOMER, … …
WHERE … … …
ORDER BY FIRST_NAME
FETCH FIRST 30 ROWS ONLY
OPTIMIZED FOR READ ONLY
```

High Performance Page Traversal for DB2 (contd.)

ObjectSource

- Next Page

```
SELECT * FROM (
SELECT FIRST_NAME, LAST_NAME, ADDRESS
FROM CUSTOMER, … …
WHERE … … …
ORDER BY FIRST_NAME
)
AS CUST_TEMP WHERE
ROW_NEXT BETWEEN 31 AND 60
OPTIMIZED FOR READ ONLY
```

High Performance Page Traversal for DB2 (contd.)

ObjectSource

- Intelligent ValueList Handler
- Intelligent Prefetch

- Pager Taglib MAY be used in conjunction with the high performance page traversal, not verified.
- Performance of Pager Taglib has not been benchmarked

Assorted Topics

ObjectSource

Questions ??

ObjectSource

Better Form and Action Handling

2 hours (Includes 1 exercise)

Offbeat: Protecting JSPs from direct access

ObjectSource

- Not linking the JSPs directly does not prevent access to it.
- If your controller has access logic and JSP does not, it is the hackers best tool to pull sensitive data
- Protect JSPs from direct access
- Some app servers provide default protection by putting them under WEB-INF
- WebLogic does not support this
- For WebLogic, create a DUMMY role and let only those with DUMMY role access the page
- Don't give anybody the DUMMY role !!!

Offbeat: Managing Struts Config

ObjectSource

- XDoclet Buffs: Don't EVER use XDoclet to manage Struts Config file
- Even though the popular media wants you to believe it
- The whole idea of struts config is to centralize and get the big picture easily.
- Splitting it in multiple source files is self-defeating
- Use a decent XML editor, Struts console or in Eclipse
- If your XML is complicated and large, think about creating multiple Modules

DispatchAction

ObjectSource

- /showCustomerForm.do?action=Create
- /showCustomerForm.do?action=Edit
- ShowCustomerAction has the following:
  - if(action.equals("Create") { ..}
    else if (action.equals("Edit") { ..}
- OR you can use DispatchAction

```
ShowCustomerAction extends DispatchAction {
    Create(ActionMapping, …) { …}
    Edit(ActionMapping, …) { …}
}
```

- One method for each action=XYZ

Multipage Forms

ObjectSource

- Multiple JSPs contain and collect data related to a single form
- Best Practice: Use one Action per JSP, not one per ActionForm

Creating Base Form & Action

ObjectSource

- Applies more to Action
- To capture & centralize boilerplate code for applications
- Log entry and exits
- Capture top level exceptions and log
- Create other second level customization to Struts
- First level customization is rolling your own RequestProcessor, ActionForward etc.
- Even if you don't have anything to centralize, create a Base Action. You will need it later !!

DispatchAction and Base Action

ObjectSource

- DispatchActions are not Base Action friendly (since they directly invoke the method)
- You will need a separate Base Action for DispatchActions
- Don't use DispatchAction if you don't absolutely need it

Dynamic Action Forms

ObjectSource

- DynactionForms - Dynamic Action Forms that don't need Java class for its Form.
- Instead, you can declare it in XML as follows:

```
<form-bean name="CustomerForm"
            type="org.apache.struts.action.DynaActionForm">
 <form-property name="firstName" type="java.lang.String"/>
 <form-property name="lastName" type="java.lang.String"
                                    initial="Doe"/>
</form-bean>
```

Dynamic Action Forms (contd.)

ObjectSource

Offbeat:

- Having a lesser class to deal with sounds like a exciting idea. However
  - Not strongly typed. No compile time checks. Need to cast fields in Action
  - How is it different from request.getParameter()
  - It bloats your struts config file (Java packages is a better alternative)
  - Time savings .. None at all
- Don't make it part of your final deployment

Dynamic Action Forms (contd.)

ObjectSource

- Good for prototyping
- Good for isolating UI design from Java code
- Combine XML based Form with ForwardAction during prototyping

Exception Handling

ObjectSource

- When Exception occurs,
  - Generate a meaningful error message for the user
  - Generate a unique id to track the error
- System.out.println does not work in production environment
  - No console in production
  - They are expensive
- Use Log4J or JDK Logging
- Wrap either with Commons Logging

Exception 101

ObjectSource

- Two types: Checked and Unchecked
- If you can't handle (or add value), don't catch the Exception
- Catch exception as close to the source
- When you catch it, log it
- Don't EVER write a do nothing catch()..
- Preserve the content, if the intent is unknown
- Use Typed Exceptions for application exceptions
- Subclass Runtime Exception for System exceptions or wrap them in application exception

Exceptions and Servlet Specification

ObjectSource

- Servlet and Filters are the catch all points in the web application

doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

- Implies the web application can only throw
  - ServletException, IOException & subclasses
  - RuntimeException and subclasses
- All other exceptions are caught & logged OR wrap it in ServletException & rethrow

Exceptions and Servlet Specification (contd.)

ObjectSource

- For ServletException, IOException, RuntimException & their subclasses:

```
<error-page>
  <exception-type>UnhandledException</exception-type>
  <location>UnhandledException.jsp</location>
</error-page>
```

- JSP Tags can throw JSPException & is not a subclass of Servlet or IOException
- Use <%@ page errorPage="/error.jsp" %> in JSPs

Exceptions and Servlet Specification (contd.)

ObjectSource

- Struts Exception Handling mechanism can handle for any exception without limitations
- When used to handle non-Servlet, non-IO Exceptions, it is complementary to Servlet specification

Struts and Exception Handling

ObjectSource

- CustomerControllerService throws DuplicateCustomerException if trying to add an existing customer
- In Struts, you can do this:

```
<exception key="database.error.duplicate"
           path="/UserExists.jsp"
           type="DuplicateCustomerException"/>
```

- Exception declaration can be local to a ActionMapping or global to the entire config file
- Uses a default exception handler

Advanced Struts Exception Handling

ObjectSource

- Cannot do Value replacement in default Struts exception handling
- Roll your own handler in such cases

```
<exception type="DuplicateCustomerException“
                handler=“DupExceptionHandler”/>
```

- Implement execute() method in ExceptionHandler

Best Practices for Struts and Exception Handling

ObjectSource

- Centralize controller based exception handling within Struts config file (applies to system exceptions)
- Application exceptions are better shown as validation errors that can be corrected before proceeding
- If System exceptions have a hierarchy, then a single global exception declaration in struts-config.xml can handle all of them.
- Use web.xml exception handling (Error Page declaration) to deal with container infrastructure errors and Tag related errors

Duplicate Form submissions

ObjectSource

- Can happen due to
  - Page Refresh
  - Browser back button and resubmit
  - Browser History feature and resubmit
- Sometimes acceptable or may be validated
- Sometimes it has to be prevented at all cost
- Struts has three ways to handle Duplicate Form submissions
  - Action Chaining
  - Redirect
  - Synchronizer Token

Duplicate Form submissions

ObjectSource

- Preventing duplicate form submisions
  - Redirect
  - Action Chaining
  - Token

Synchronizer Tokens

ObjectSource

- Action class has a saveToken() method
- Puts a unique random id in the session
- Add a hidden field to the html form to hold the token
- In the Action that handles Form submission, use the isTokenValid() method to check the token
- It checks if the token in session is same as token in form submission.
- If they are not, handle the duplicate submission
- If they are same, the token is removed from session

Better Form and Action Handling

ObjectSource

Questions ??

ObjectSource

Struts Validation

1 hour (Includes 1 exercise)

Commons Validator

ObjectSource

- Jakarta Commons sub project
- Started in 2000
- Yet to reach maturity in some areas
- AdvanTGWeb Rule Services is much feature rich
- Opinion is biased ☺
- Struts comes with built-in integration for Commons Validator

Basics

ObjectSource

- `GenericValidator` - Basic validator class in Commons Validator
- Contains Fine grained validation routines such as `isFloat()`, `isBlankOrNull()`
- Commons Validator is pure Java no dependency on J2EE
- `FieldChecks` – A Struts class that couples Commons Validator to Struts
- Defines coarse grained methods such as validateRequired(), validateDate(), validateCreditCard() etc.

Struts and Validator

ObjectSource

- Contains two xml files
- Generally named validation-rules.xml & validation.xml
- validation-rules.xml contains reusable rules
- validation.xml binds those rules to Struts Forms

validation-rules.xml

ObjectSource

```
<form-validation>
<global>
    <validator name="required“ classname=“FieldChecks"
      method="validateRequired“ methodParams="Object,
      ValidatorAction,Field,ActionErrors,HttpServletRequest"
      msg="errors.required">
    </validator>
... ...
</global>
</form-validation>
```

In simple english

ObjectSource

- The Rule named “required” is defined in a method validateRequired() in FieldChecks and it takes the specified parameters and validates the first Object passed in. If the validation fails, it throws the error message corresponding to errors.required key in the message resources

Another example

ObjectSource

- Validation rules can be dependent on others

```
<validator name="minlength“ classname="FieldChecks"
    method="validateMinLength"
    methodParams="Object,ValidatorAction,
                  Field,ActionErrors,HttpServletRequest"
    depends="required"
    msg="errors.minlength">
</validator>
```

We move on to validation.xml

ObjectSource

```
<form-validation>
  <formset>
    <form name="CustomerForm">
      <field property="firstName“ depends="required">
          <arg0 key="prompt.customer.firstname"/>
      </field>
    </form>
  </formset>
</form-validation>
```

validation.xml – Multiple rules

ObjectSource

```
<form-validation>
<formset>
<form name="CustomerForm">
 <field property="firstName“ depends="required,minlength">
    <arg0 key="prompt.customer.firstname"/>
    <arg1 name="len" key="1" resource="false"/>
 </field>
</form>
</formset>
</form-validation>
```

Message Resources

ObjectSource

- Upto 4 args can be added to validation
- Corresponds to four value replacement variables
- Add error messages with keys defined in validation-rules.xml
- errors.required=(0) is required
- errors.minlength={0} cannot be less than {1} characters

validation.xml - Variables

ObjectSource

```
<form name="CustomerForm">
 <field property="firstName“ depends="required,minlength">
    <arg0 key="prompt.customer.firstname"/>
    <arg1 name="len" key="${var:minlen}" resource="false"/>
    <var>
          <var-name>minlen</var-name>
          <var-value>1</var-value>
    </var>
 </field>
</form>
```

Validator Configuration

ObjectSource

- Configured as a Struts Plugin
- Struts Plugin is configured in struts-config.xml
- Lifecycle managed by Struts
- Two methods – init and destroy

```
<plug-in className="org.apache….ValidatorPlugIn">
    <set-property property="pathnames“
                  value="/WEB-INF/validator-rules.xml,
                         /WEB-INF/validation.xml"/>
</plug-in>
```

ValidatorForm

ObjectSource

- Subclass of ActionForm
- Already implements validate() method
- Extend your ActionForm from ValidatorForm & you are ready to go

DynaValidatorForm

ObjectSource

- ValidatorForm equivalent for DynaActionForms

```
<form-bean name="CustomerForm“
 type="org.apache.struts.validator.DynaValidatorForm">
    <form-property name="firstName" type="String "/>
    <form-property name="lastName" type="String"
    initial="Doe"/>
</form-bean>
```

Validation & Multipage forms

ObjectSource

```
<form name="CustomerForm">
<field property="firstNm" page="1" depends="required">
   <arg0 key="customerform.firstname"/>
</field>
<field property="fieldX" page="2" depends="required">
   <arg0 key="customerform.fieldX"/>
</field>
</form>
```

Struts Validation

ObjectSource

Questions ??

ObjectSource

Struts Modules

1 hour (Includes 1 exercise)

Struts Config life in a Project

ObjectSource

- Month 1: You start with a fresh neat xml
- Month 2: Its growing but no worries…
- Month 4: Its becoming unmanageable
- Month 6: Cant take this anymore….

Modules

ObjectSource

- It's not a web application
- A Logical division within the application
- Like rooms in a house
- They share the same libraries, same web.xml
- But have a struts-config.xml each
- Each struts-config.xml has its own RequestProcessor, MessageResources etc.
- Helps upgrade to new features easier

web.xml for multiple modules

ObjectSource

```
<servlet>
<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>config/module1</param-name>
    <param-value>/WEB-INF/struts-m1-config.xml</param-value>
</init-param>
</servlet>
```

Another way of using html:link

ObjectSource

- <html:link forward="gotox">Go to X</html:link>
- Can map to this in struts-config.xml:

```
<global-forwards>
    <forward name="gotox" path=".." />
</global-forwards>
```

- The path can be a JSP or a ActionMapping

Navigating between Modules

ObjectSource

- All urls in struts-config.xml relative to the module
- How to navigate between modules?
- SwitchAction to the rescue
- Define SwitchAction as:
- <action path="/switch"
  type="org.apache.struts.actions.SwitchAction"/>

Navigating between Modules (contd.)

ObjectSource

- In the top level default module, define navigations to other modules as global forwards

```
<forward name="showCust"
        path="/switch.do?page=/showCustomerForm.do
              &amp;prefix=/customerModule" />
```

- In the JSPs for the top level module define links as:

```
<html:link forward="showCust">Go to XYZ</html:link>
```

- Repeat the same for all inter-module navigation
- Never cross modules for form submission !!

Struts Modules

ObjectSource

Questions ??

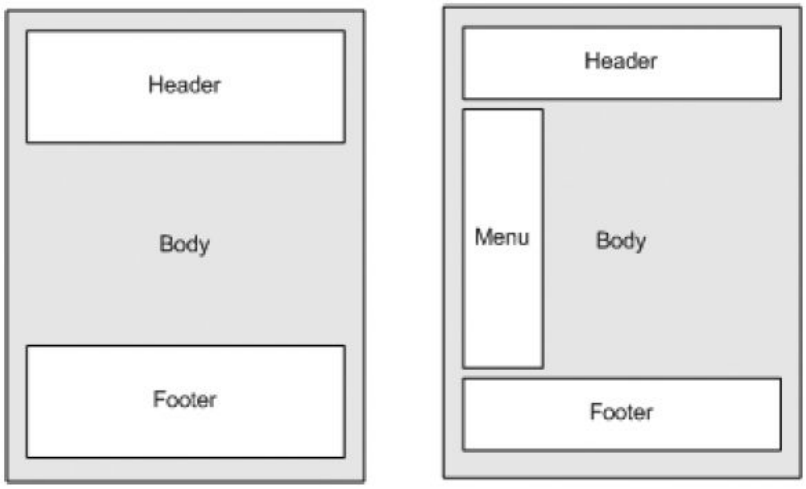
ObjectSource

Struts and Tiles

1 hour (Includes 1 exercise)

Page layout

ObjectSource



About headers and footers

ObjectSource

- Can be hardcoded in every JSP
- Better Option: Can use <jsp:incude>
- Problems with jsp:include
  - Layout change affects every page
- Best Option: Use Templating

Tiles

ObjectSource

- Tiles is a templating framework
- <jsp:include> turned inside out
- Very easy and robust
- Basic principle behind Tiles: Refactor the layout common to JSPs to a higher level and then reuse it across them
- Layout is a JSP by itself
- Layout represents the entire displayed page.
- Layout defines areas in the page
- Other JSPs are plugged into the areas dynamically

Layout JSP

ObjectSource

```
<html>
<body>
<TABLE border="0" width="100%" cellspacing="5">
  <tr><td><tiles:insert attribute="header"/></td></tr>
  <tr><td><tiles:insert attribute="body"/></td></tr>
  <tr><td><hr></td></tr>
  <tr><td><tiles:insert attribute="footer"/></td></tr>
</TABLE>
</body>
</html>
```

Tiles Definition

ObjectSource

- Where are header and footer defined?

```
<definition name="/cust.page" path="/Layout.jsp">
  <put name="header" value="/comm/header.jsp" />
  <put name="footer" value="/comm/footer.jsp" />
  <put name="body" value="/CustomerForm.jsp" />
</definition>
```

TilesRequestProcessor

ObjectSource

- A different RequestProcessor for Tiles
- Includes JSP responses into committed output stream

```
<action path="/showCustomerForm“
        type="ShowCustomerAction">
    <forward name="success" path="/cust.page"/>
</action>
```

Tiles and Inheritance

ObjectSource

- Layout can be reused and values overridden like OO
- Layout can be extended like OO

Tiles Best Practices

ObjectSource

- Create Base Definition for each layout style
- Define common headers and footers in the base definition
- Extend the base definition for individual pages and just add body definition

Tiles Configuration

ObjectSource

- Add Tiles Plugin in struts-config.xml
- Tiles PlugIn loads the Tiles Definition file
- Add TilesRequestProcessor to struts-config.xml

Struts and Tiles

ObjectSource

Questions ??