**Experiment No. 2**

**Title: Write an application to design form using struts2 UI component to accept and validate details of student using ActionSupport.**

**Batch:B1 RollNo.:1514033 ExperimentNo.:2**

### Aim: Write an application to design form using struts2 UI component to accept and validate details of student using ActionSupport.

### --------------------------------------------------------------------------------------------

**Resources needed:**Eclipse,Tomcat,Struts2.

### Theory:

Struts 2 Action

In struts 2, action class is POJO (Plain Old Java Object).POJO means you are not forced to implement any interface or extend any class.Generally, execute method should be specified that represents the business logic.

Action Interface

A convenient approach is to implement the **com.opensymphony.xwork2.Action** interface that defines 5 constants and one execute method.Action interface provides 5 constants that can be returned form the action class. They are:

* SUCCESS indicates that action execution is successful and a success result should be shown to the user.
* ERROR indicates that action execution is failed and a error result should be shown to the user.
* LOGIN indicates that user is not logged-in and a login result should be shown to the user.
* INPUT indicates that validation is failed and a input result should be shown to the user again.
* NONE indicates that action execution is successful but no result should be shown to the user.

Action interface contains only one method execute that should be implemented overridden by the action class even if you are not forced.

public String execute();

Action Support class and its interfaces

ActionSupport is an XWork class that provides default implementations of many interfaces.

ActionSupport implements the following six interfaces:

* Action
* Validateable
* ValidationAware
* TextProvider
* LocaleProvider
* Serializable

ValidationAware interface

The ValidationAware interface can accept the field level or action class level error messages. The field level messages are kept in Map and Action class level messages are kept in collection. It should be implemented by the action class to add any error message. The methods of ValidationAware interface are as follows:

|  |  |
| --- | --- |
| **Method** | **Description** |
| **void addFieldError(String fieldName,StringerrorMessage)** | adds the error message for the specified field. |
| **void addActionError(String errorMessage)** | adds an Action-level error message for this action. |
| **void addActionMessage(String message)** | adds an Action-level message for this action. |
| **void setFieldErrors(Map<String,List<String>> map)** | sets a collection of error messages for fields. |
| **void setActionErrors(Collection<String>errorMessages)** | sets a collection of error messages for this action. |
| **void setActionMessages(Collection<String> messages)** | sets a collection of messages for this action. |
| **booleanhasErrors()** | checks if there are any field or action errors. |
| **booleanhasFieldErrors()** | checks if there are any field errors. |
| **booleanhasActionErrors()** | checks if there are any Action-level error messages. |
| **booleanhasActionMessages()** | checks if there are any Action-level messages. |
| **Map<String,List<String>>getFieldErrors()** | returns all the field level error messages. |
| **Collection<String>getActionErrors()** | returns all the Action-level error messages. |
| **Collection<String>getActionMessages()** | returns all the Action-level messages. |

More complicated validation should probably be done in the validate() method.When the user presses the submit button, Struts 2 will transfer the user's input to the instance fields. Then Struts 2 will automatically execute the validate method. If any of the if statements are true, Struts 2 will call its addFieldError method (which our Action class inherited by extending ActionSupport).

If any errors have been added then Struts 2 will not proceed to call the execute method. Rather the Struts 2 framework will return "input" as the result of calling the action.

Steps to perform custom validation

The steps are as follows:

1. **create the form to get input from the user**
2. **Define the validation logic in action class by extending the ActionSupport class and overriding the validate method**
3. **Define result for the error message by the name input in struts.xml file**

### Results: (Screen shots of application development steps, program code and web browser displaying the specified message.)

### Index.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<h1>Welcome</h1><br>

<a href=*"login"*>Login</a>

</body>

### </html>

### login.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<s:form action=*"loginprocess"*>

<s:textfield name=*"username"* label=*"Name"*></s:textfield>

<s:password name=*"password"* label=*"Password"*></s:password>

<s:submit value=*"login"*></s:submit>

</s:form>

</body>

### </html>

### loginprocess.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

HELLO <s:property value=*"username"*/>

<s:form action=*"registration"*>

<s:textfield name=*"name"* label=*"Name"*></s:textfield>

<s:textarea name=*"address"* label=*"Project Description"*></s:textarea>

<s:radio name=*"gender"* label=*"Gender"* list=*"{'Male','Female'}"*></s:radio>

<s:select label=*"Select your branch"*

headerKey=*"-1"* headerValue=*"Select Branch"*

list=*"*#{'1':'IT','2':'Comp','3':'Mech','4':'Extc','5':'Etrx'}*"*

name=*"yourBranch"*

value=*"3"*></s:select>

<s:select name=*"year"* list=*"{'FE','SE','TE','BE'}"*

value=*"year"*

label=*"Year of study"*

multiple=*"true"*></s:select>

<s:checkboxlist label=*"Tick appropriate project category"*

list=*"{'Android Application','Hardware based','Web Application'}"*

name=*"pcategory"*

vale=*"Android Application"*></s:checkboxlist>

<s:checkbox name=*"reportstatus"* value=*"true"* label=*"Have you submitted your report"*></s:checkbox>

<s:submit value=*"Submit"*></s:submit>

</s:form>

</body>

### </html>

### display.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<%@ taglib uri=*"/struts-tags"* prefix=*"s"* %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

Hello <s:property value=*"name"*/><br>

Description: <s:property value=*"address"*/><br>

Gender: <s:property value=*"gender"*/><br>

Branch: <s:property value=*"yourBranch"*/><br>

Year: <s:property value=*"year"*/><br>

Project: <s:property value=*"pcategory"*/><br>

Reportstatus: <s:property value=*"reportstatus"*/>

<br>

</body>

### </html>

### struts.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<package name=*"default"* extends=*"struts-default"*>

<action name=*"login"*>

<result>login.jsp</result>

</action>

<action name=*"loginprocess"* class=*"com.Action.LoginAction"*>

<result name=*"success"*>loginprocess.jsp</result>

<result name=*"input"*>login.jsp</result>

</action>

<action name=*"registration"* class=*"com.Action.Register"*>

<result name=*"success"*>display.jsp</result>

<result name=*"input"*>loginprocess.jsp</result>

</action>

</package>

### </struts>

### LoginAction.java

**package** com.Action;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** LoginAction **extends** ActionSupport {

**private** String username;

**private** String password;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** **void** validate()

{

**if**(username.isEmpty())

addFieldError("username","Username cannot be blank");

**if**(password.isEmpty())

addFieldError("password","Password cannot be blank.");

}

**public** String execute(){

**if**(username.equals("admin") &&password.equals("0000"))

**return** "success";

**else** **if**(username.equals("admin")){

addFieldError("password","Incorrect Password");

**return** "input";

}**else**{

addFieldError("username","Incorrect Username");

**return** "input";

}

}

}

### Register.java

**package** com.Action;

**import** com.opensymphony.xwork2.ActionSupport;

**public** **class** Register **extends** ActionSupport{

**private** String name,address,gender,yourBranch,year;

**private** String pcategory[]=**new** String[10];

**private** **boolean** reportstatus;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getGender() {

**return** gender;

}

**public** **void** setGender(String gender) {

**this**.gender = gender;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getYourBranch() {

**return** yourBranch;

}

**public** **void** setYourBranch(String yourBranch) {

**this**.yourBranch = yourBranch;

}

**public** **boolean** getReportstatus() {

**return** reportstatus;

}

**public** **void** setReportstatus(**boolean** reportstatus) {

**this**.reportstatus = reportstatus;

}

**public** String getYear() {

**return** year;

}

**public** **void** setYear(String year) {

**this**.year = year;

}

**public** String execute(){

**return** "success";

}

**public** String[] getPcategory() {

**return** pcategory;

}

**public** **void** setPcategory(String pcategory[]) {

**this**.pcategory = pcategory;

}

}

### Register-validation.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE validators PUBLIC "-//Apache Struts//XWork Validator 1.0.3//EN" "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">

<validators>

<field name=*"name"*>

<field-validator type=*"requiredstring"*>

<message>Name must not be blank.</message>

</field-validator>

</field>

<field name=*"address"*>

<field-validator type=*"stringlength"*>

<param name=*"maxLength"*>100</param>

<param name=*"minLength"*>8</param>

<message>Description should be between 8 to 100 char</message>

</field-validator>

</field>

<field name=*"pcategory"*>

<field-validator type=*"fieldexpression"*>

<param name=*"expression"*><![CDATA[pcategory.length> 1]]></param>

<message>Select atleast two category.</message>

</field-validator>

</field>

<field name=*"yourBranch"*>

<field-validator type=*"fieldexpression"*>

<param name=*"expression"*><![CDATA[yourBranch!=-1]]></param>

<message>Select branch.</message>

</field-validator>

</field>

<field name=*"year"*>

<field-validator type=*"fieldexpression"*>

<param name=*"expression"*><![CDATA[year!="year"]]></param>

<message>Select Year.</message>

</field-validator>

</field>

<field name=*"gender"*>

<field-validator type=*"required"*>

<param name=*"trim"*>true</param>

<message>Mark gender</message>

</field-validator>

</field>

<field name=*"reportstatus"*>

<field-validator type=*"required"*>

<message>Report should be submitted</message>

</field-validator>

</field>

</validators>

### Output:

### 

### 

### 

### 

### 

### 

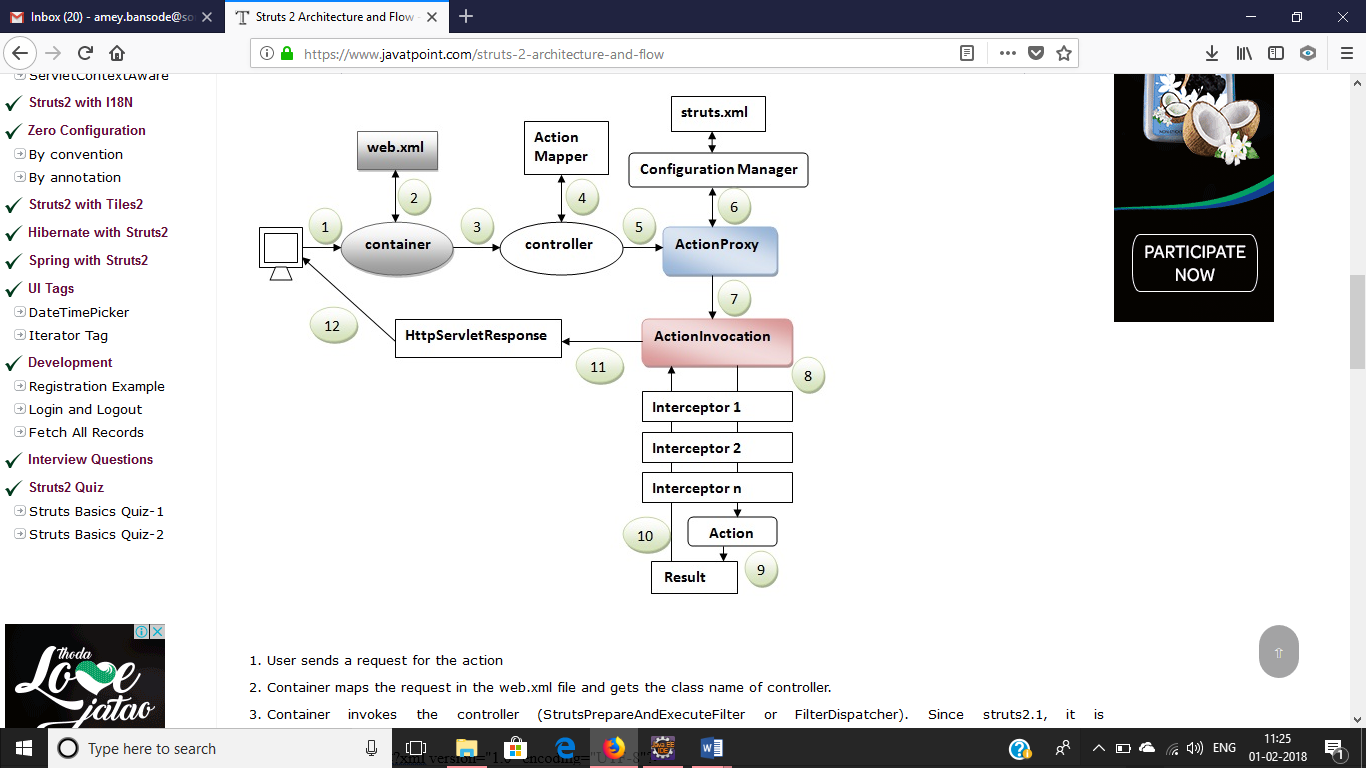
### 

### 

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Questions:**

1. Explain the flow of execution for struts program.



1. User sends a request for the action
2. Container maps the request in the web.xml file and gets the class name of controller.
3. Container invokes the controller (StrutsPrepareAndExecuteFilter or FilterDispatcher). Since struts2.1, it is StrutsPrepareAndExecuteFilter. Before 2.1 it was FilterDispatcher.
4. Controller gets the information for the action from the ActionMapper
5. Controller invokes the ActionProxy
6. ActionProxy gets the information of action and interceptor stack from the configuration manager which gets the information from the struts.xml file.
7. ActionProxy forwards the request to the ActionInvocation
8. ActionInvocation invokes each interceptors and action
9. A result is generated
10. The result is sent back to the ActionInvocation
11. A HttpServletResponse is generated
12. Response is sent to the us

Outcomes: Develop Applications using Frameworks

**Conclusion: (Conclusion to be based on the objectives and outcomesachieved)**

We have successfully developed an application to design form using struts2 UI component to accept and validate details of the different UI components.

**Grade: AA / AB / BB / BC / CC / CD/DD**

**Signature of faculty in-charge withdate**

**References:**

1.By James Holmes ; ”Struts: The Complete Reference”; 2nd Edition ; McGraw-Hill

Publication

2.ByChuckCavaness; “Programming Jakarta Struts” ; 2nd Edition ; O'Reilly Media

Publication; 2004