24/10/13

STRUTS

Course Title :- Struts (1.x & 2.x)

Rs = 85/=

Pre-requisite :- Servlets, JSP.

Course objects :- java web application development.

Q) what is not Struts ?

⇒ Struts is not a web technology unlike servlets & JSP.

⇒ Struts is neither J2SE nor J2EE (but uses both).

⇒ Struts is not a specification unlike servlets, JSP, JDBC etc

⇒ Struts is not meant for replacing Servlets or JSP (instead it complements them).

Q) what is a struts application ?

⇒ A java web application that is developed using Struts is nothing but a Struts Application.

Q) what are the java technologies (J2EE/J2SE) used in a Struts application ?

⇒ 1) Servlets

2) JSP

3) JavaBeans.

Q) what is Struts ?

⇒ Struts is an open source, java web application framework from ASF.

⇒ Struts is a software product to be Installed into a computer system inorder to develop java web application powered by struts.

Q) What is the purpose of Struts?

⇒ Struts is used to develop MVC based java web application MVC (model view controller).

Q) Can we develop a java web application that follows MVC architecture without Struts?

⇒ yes
⇒ Servlet technology for controllers development.
⇒ JSP technology for view components development.
⇒ Java Beans for model components development.
⇒ Developing controller components and view components only with Servlet technology & JSP technology support is complex.

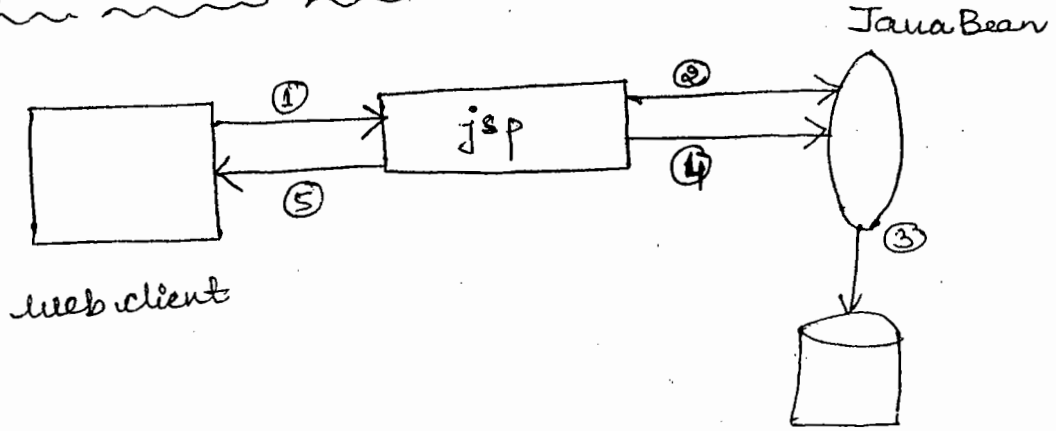
Rapid Application development is not promoted. As a result, application development is not cost effective.

Q) Explain about JSP architectural models suggested by SUN Microsystem for Java web application development?

⇒ Sun Microsystem suggested 2 JSP architectural models for java web application development.

i) JSP model-I architecture (page centric Architecture)
ii) JSP model-II architecture (MVC architecture)

① JSP model-I architecture :-



① JSP receiving the client request and capturing the user I/P.

② JSP invoking java bean for data processing or processed data.

③ Java bean accessing the data from the database. Java bean processes the data & hold the processed data.

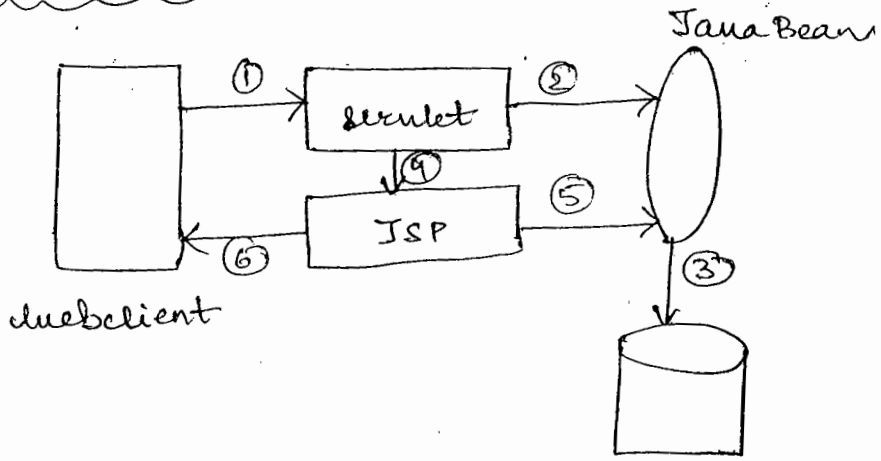
④ JSP accessing the processed data from java bean.

⑤ JSP producing the response paged for the client.

note :-
In this architectural style of application development only data accessing and processing and separated from the web component.

i.e :- jsp still in a JSP flow control code and O/P generation code mixed. Therefore, this architecture not suitable for large scale applications.

② JSP-model-II architecture :-



① Servlet receiving the client request and capturing with user I/P.

② Servlet invoking the java bean or processed data.

③ Java bean accessing the data from database. Java bean processes the data and holds the processed data.

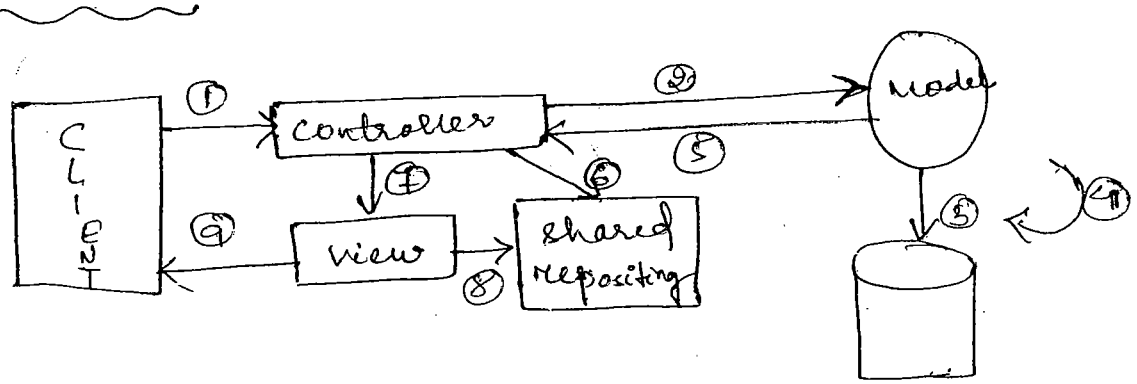
④ Servlet switching the control to JSP.

⑤ JSP accessing the processed data from java bean.

⑥ JSP producing the response page for the client.

note :-
with slite modification MVC architecture only currently followed in the industry for java web application development.

MVC architecture :-



① Controller receiving client request and capturing the user I/P.

② Controller invoking the business method of business component (model component) for processed the data.

③ Accessing data from the database.

④ model component processing the data according to the business rules of the business organisation.

⑤ model component returning processed data to the controller.

⑥ controller storing the data into the scope.

⑦ controller switching the control to view component.

⑧ View component returning processed data from the scope.

⑨ View component processing the response for the client.

Q) what is the benefits of MVC?

Ans) → In MVC architecture style of web application development, three concerns are clearly separated.

a) code that generates the o/p for the end-users.
b) code that takes care of work flow of the application.
c) code that process data (after accessing from database)

→ such clear separation of concerns gives the following benefits.

(i) parallel development.
(ii) Code optimisation.
(iii) clear division of rules
(iv) better code maintainance.
(v) without affecting one concern, the other concern can be modified.

(vii) Code generation tools can be effectively used the promotes RAD.

(viii) Better productivity.

(ix) cost effectiveness in application developer.

(x) Code reuse.

28/10/13

Q) What is open source software?

⇒ A software is said to be open source software if its sourcecode is publicly available.

for eg. linux OS, Struts, spring & hibernate.

⇒ closed source software's source code is not available publicly.

for eg. Windows OS.

Q) What is a java application framework?

⇒ An application framework is a reusable semifinished application that can be customised to develop any specific application.

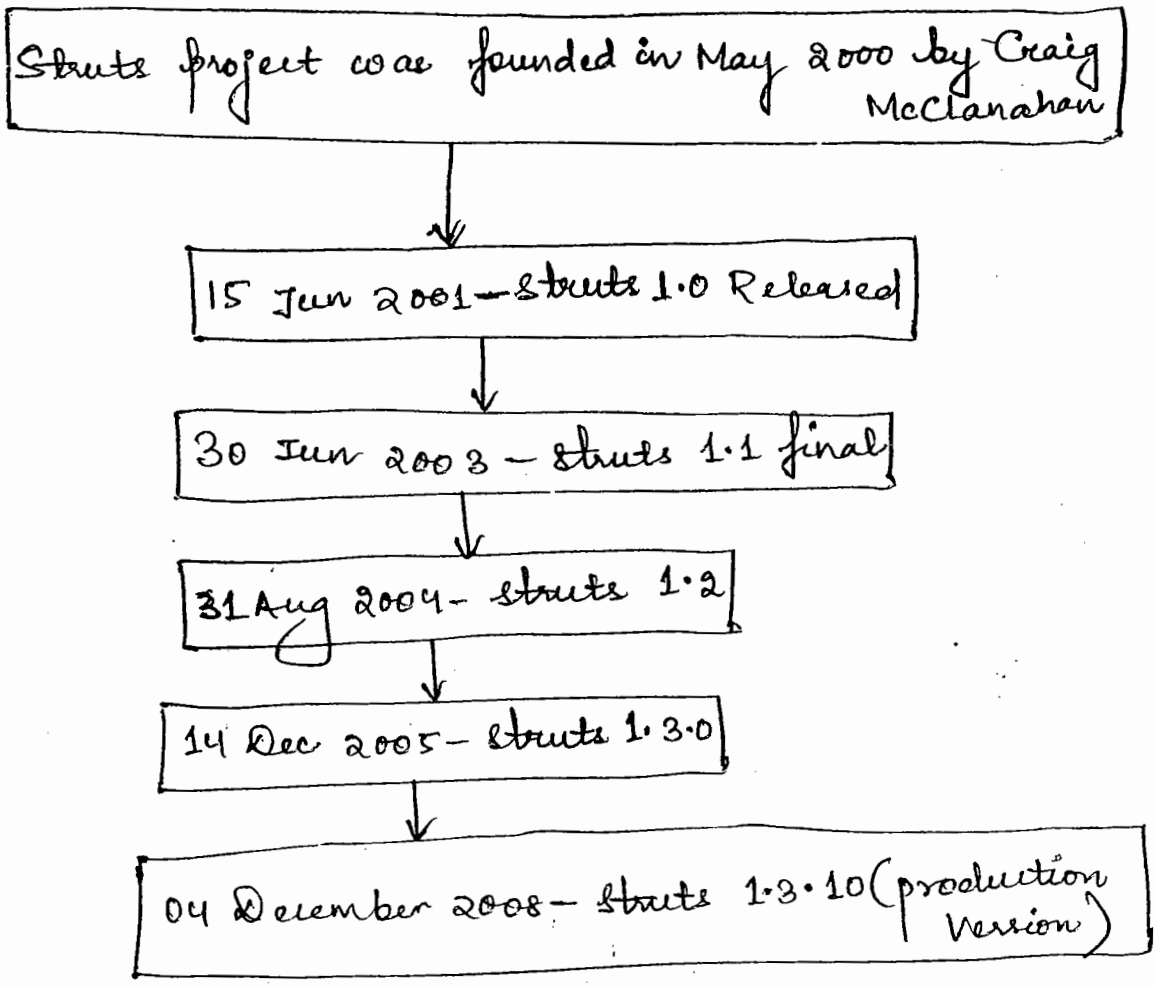
⇒ A java application framework is written in java and is used in java enterprise application development.

⇒ An application framework is meant for reducing the complexity involved in each partition(layer) of java enterprise application. for eg. Struts in front-end part (presentation layer), Hibernate in back-end communication (data access layer).

⇒ A java framework is a tool kind of software which is a collection of java classes and interfaces that provides infrastructural services to java applications.

⇒ framework invokes application code & is proactive in providing the service to the application. where as, API is reactive and it doesn't call your application code.

Q) Give a note on Evolution of Struts?



29/10/13

Q) What is the objective of Struts?

⇒ Developing presentation layer for web-enabled (online) java enterprise application.

⇒ Struts therefore is known as UI framework or presentation layer framework.

⇒ Being presentation layer framework i.e front-end framework, Struts supports controller and view components development.

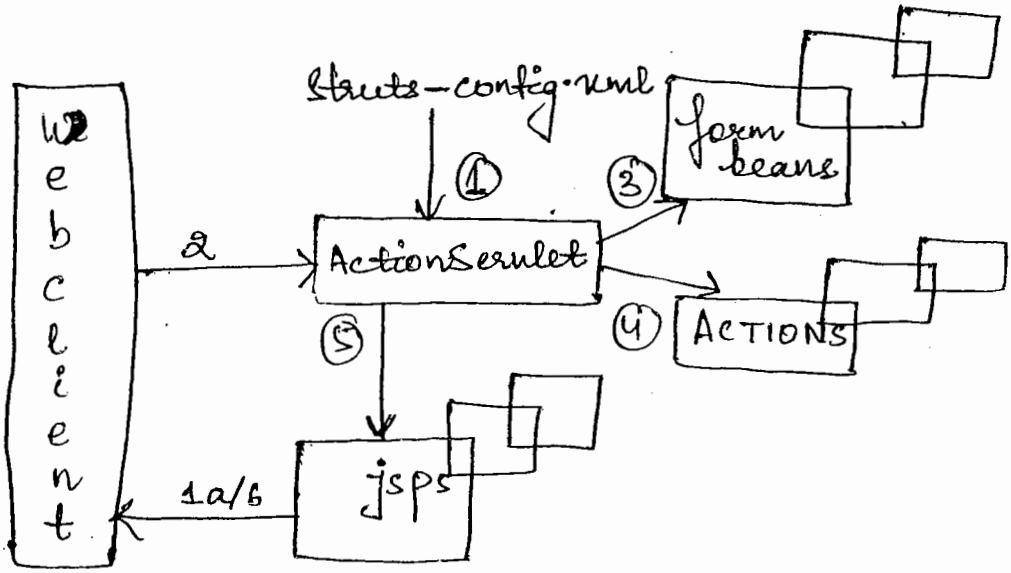
⇒ Struts doesn't provide any support in model component (business component) development.

Q) what is Struts architecture ?

⇒ Struts framework is meant for MVC based java web application development.

⇒ There are 5 architectural elements in struts.

1) ActionServlet
2) form beans
3) actions
4) struts-config.xml.
5) jsps

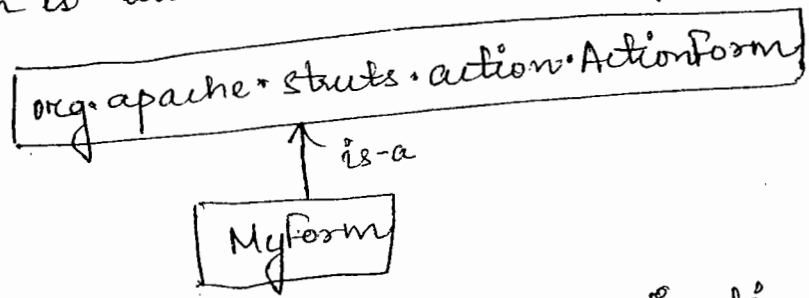


30/10/13

Q) Explain about form beans ?

⇒ A form bean is a user defined java class that extends org.apache.struts.action.ActionForm.

⇒ A form bean is also known as action form.



⇒ for each* web form of the Struts application we develop an action form and it is a Java bean. Therefore, it is known as a form bean.

⇒ for each input field of the web form we have one instance variable in form bean.

⇒ request parameter names of the form fields and form bean fields' name must be the same.

for eg.

```
public class LoginForm extends ActionForm
{
  private String username;
  private String password;
// setter & getter methods
} // form bean (action form)
```

⇒ a form bean is a data container that holds the user input (view data) that is heading towards model via controller.

⇒ form bean is not a POJO ~~class~~ (Plain old Java Object) class. Therefore, it can't be used as data transfer object.

note:- A java class is said to be a POJO if and only if it doesn't inherit from framework specific or technology specific interface or class.

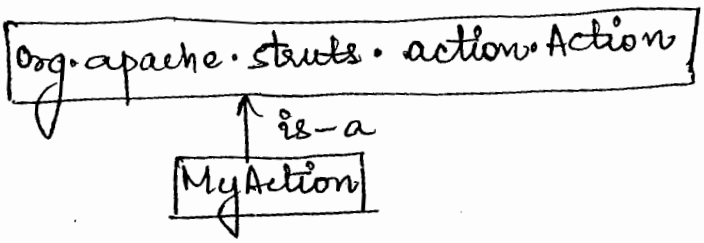
⇒ framework implicitly instantiates form bean and populates its fields with user input.

⇒ In a formbean, validate() and reset() methods also may be implemented in addition to accessors (getters) and mutators (setter).

Q) Explain about actions ?

⇒ A user defined java class that extends org.apache.struts.action.Action is an action class.

⇒ In general, for each use-case of the application one action class is to be developed by the struts developer.



⇒ for eg.

```
public class LoginAction extends Action
 {
   ..... execute(p1, p2, request, response)
   {
     .....
   }
 }
```

⇒ action class performs 2 duties majorly.

1) calling the model component method for data processing. (business).

2) returning view mapping information to ActionServlet (information about to which jsp to switch the control).

⇒ action class is a sub-controller. It contains use-case specific flow control code. i.e In a struts application, action class is a controller component.

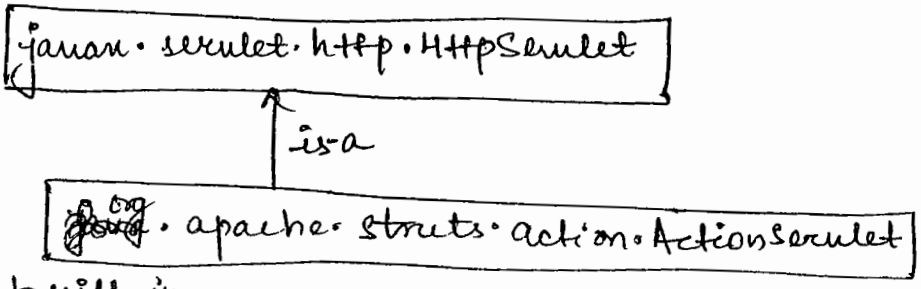
⇒ action class is a singleton (without implementing single design pattern).

⇒ only one action instance handles any number of client requests. Therefore, it is not threadsafe.

31/10/13

Q) Explain about ActionServlet ?

Ans) ⇒ ActionServlet is a HttpServlet.



⇒ It is a built-in servlet provided by Struts Framework. i.e developer need not develop it.

⇒ Struts Developer is responsible for 3 things in the Struts application in connection with ActionServlet.

1) registering it with the container.
2) pre-initializing it.
3) making it the front controller.

⇒ ActionServlet contributes to struts Application in 2 phases

⇒ During initialization phase ActionServlet reads struts-config.xml into memory and gets flow control information of all the use-cases of the struts application.

⇒ During servicing phase, it creates form bean instance & populates its fields with user input, invokes validation code if any, creates action class instance, invokes its execute method and based on the view mapping information given by action class, switches the control to the appropriate jsp using forward mechanism of the request dispatcher.

Note:- ActionServlet internally uses RequestProcessor to perform these activities during servicing phase.

⇒ ActionServlet is the front-controller of the Struts Application.

⇒ ActionServlet is able to read flow control information from xml file. Therefore, it is reusable.

⇒ ActionServlet, being a pre-developed class of the Struts framework can have only generic flow control logic and it can't have use-case specific flow control code. for that it depends on action classes.

Q) Explain about struts-config.xml ?

⇒ It is the configuration file of the struts application.

⇒ It is one * per struts application.

⇒ Struts developer should develop this file creating relationships between view components and controller components of every use-case.

⇒ formbeans, jsps and action classes have logical names in this file.

⇒ ActionServlet is able to perform its functionality using struts-config.xml.

⇒ logical name concept implemented in struts-config.xml only making Struts framework reusable.

01/11/13

Q) Explain about jsps of the struts application ?

⇒ jsps are the view components of the struts application. (MVC based java web application)

⇒ Struts doesn't provide any built-in jsps for the Struts application. Struts application developers only responsible to develop jsps.

⇒ Struts framework provides (custom) tag libraries that support jsp development.

⇒ (i) html library
(ii) bean library
(iii) logic library
(iv) tiles library

⇒ jsps are used in the following areas of a java based dynamic website (powered by struts).

1) homepage development
2) input pages development
3) response pages development
   a) response with user expected info.
   b) error response.

a jsp = HTML + JSP elements
                └ taglib directive.

⇒ jsps of a struts application use HyperText Markup language tags and Struts framework given JSP custom tags and "taglib directive".

⇒ A jsp, being a view component shouldn't perform the following things/ tasks.

1) recieving the client request.
2) capturing the user input.

3) communication with the database.

4) processing of data.

5) talking to model component for processed data.

6) deciding another view.

⇒ In a jsp, Java code shouldn't be directly written.

4/11/13

Elements of MVC Architecture

Q) Explain about model components?

⇒ Data processing according to the enterprise rules is performed in a model component.

⇒ programmatical implementation of business rule of the enterprise (business organisation) is nothing but business logic.

⇒ Business logic is encapsulated in the model component & therefore it is also known as business components.

⇒ In the begining days of Java enterprise application development which was Java enabled, Java beans were used as business components.

⇒ Java beans is missing with the following enterprise capabilities.

1) distributed computing
2) transaction support
3) security.

⇒ currently in the industry, business components services are available to controller components from.

1) spring beans
2) web services
3) EJB's (session beans)

note:- Struts developer is no way concerned with model component development.

⇒ MVC is a front-end (presentation layer) design pattern. It doesn't address the separation of data processing and data accessing code.

⇒ MVC design pattern addresses the separation of front-end (presentation layer) into 2 kinds of activities into two different components i.e flow control code into controller components and O/P generation code into view components.

Q) Explain about controller?

⇒ The component (object) in which, work flow logic (flow control logic) is encapsulated is known as controller.

⇒ A controller shouldn't do the following activities

1) communication with database.
2) processing of data.
3) producing any kind of view for the client.

Q) What are the duties of a controller in an MVC based java web application?

⇒ Capturing the user input.
⇒ validating the user input.
⇒ invoking the business method of the model component
⇒ identifying the appropriate view based on the response from the business method of the model component.
⇒ switching control to the view component.

05/11/13

Q) How are controllers classified?

⇒ Controllers are of 2 types in a java web application.

1) front controller

2) sub controller

note:- This classification is possible if and only front controller design pattern is implemented in addition to MVC.

Q) Explain about front controller?

⇒ A controller component that receives the client request upfront is nothing about front controller.

⇒ for any online service of the website request may come, front controller receives that request first.

⇒ only one front controller for the entire application.

⇒ front-controller doesn't have use-case specific flow control code encapsulated into it. ie front controller doesn't invoke the business method & can't identify the view (response page).

⇒ front-controllers has generic flow control code which is required in common for all the use cases of the application.

⇒ front-controller provides majorly 2 benefits.

1) reusability of generic flow control code.

2) single pt. entry into the website.

note:- In Struts 1.x applications "A servlet" acts as front-controller i.e ActionServlet.

⇒ In Struts 2.x applications "A servlet filter" act as front-controller.

Q) Explain about sub controllers?

⇒ A controller that encapsulates use-case specific flow control code is nothing but a sub controller.

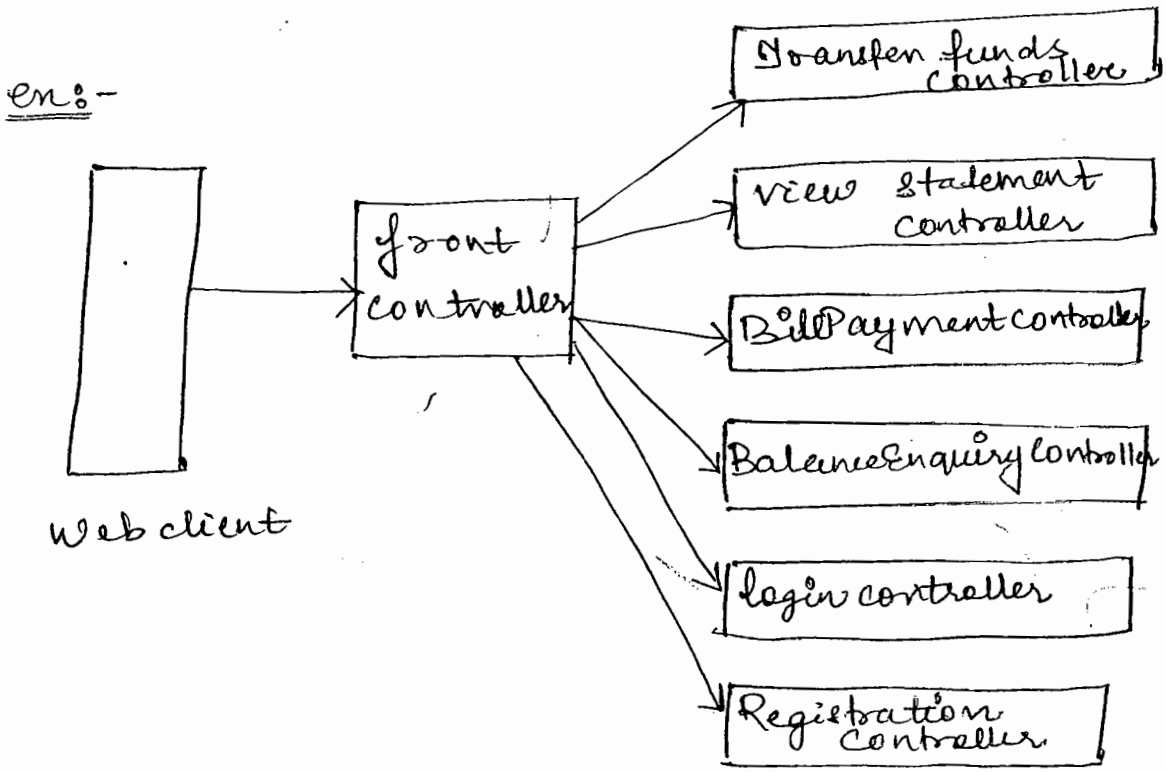
⇒ In general, one sub controller per online service.

⇒ sub controller does the following.

1) calling the business method.
2) informing the front-controller to which view to switch the control.

note:- In struts 1.x application, action class is the sub controller.

eg:-



Q) Explain about view components?

⇒ What user sees of a website is a view.

⇒ View components are those web components that produce the views of the application.

⇒ Mostly jsps are the view components.

⇒ View components are used to develop input pages & response pages.

note :- In jsps, only HTML tags and JSP tags should be used but not the scripting elements.

⇒ writing java code in jsps has following drawbacks.

1) proprietary code of the applicatn is exposed to business clients.

2) code generation tools can't generate mixed code & therefore, RAD is not promoted.

3) code is not reusable.

4) dealing with syntactical errors of java happens at application running time instead of development time.

6/11/13

Q) What are the system requirements to develop and run a struts application?

Ans) 1) Install J2SE platform.

2) Install J2EE platform (web container)

3) Struts jar files (to be placed into lib folder)

Q) what are the steps involved in a Struts Application development?

Step1:- create a structured heirarchy of directories.

Step2:- develop the jsps.

Step3:- develop the form beans.

Step4:- develop action classes.

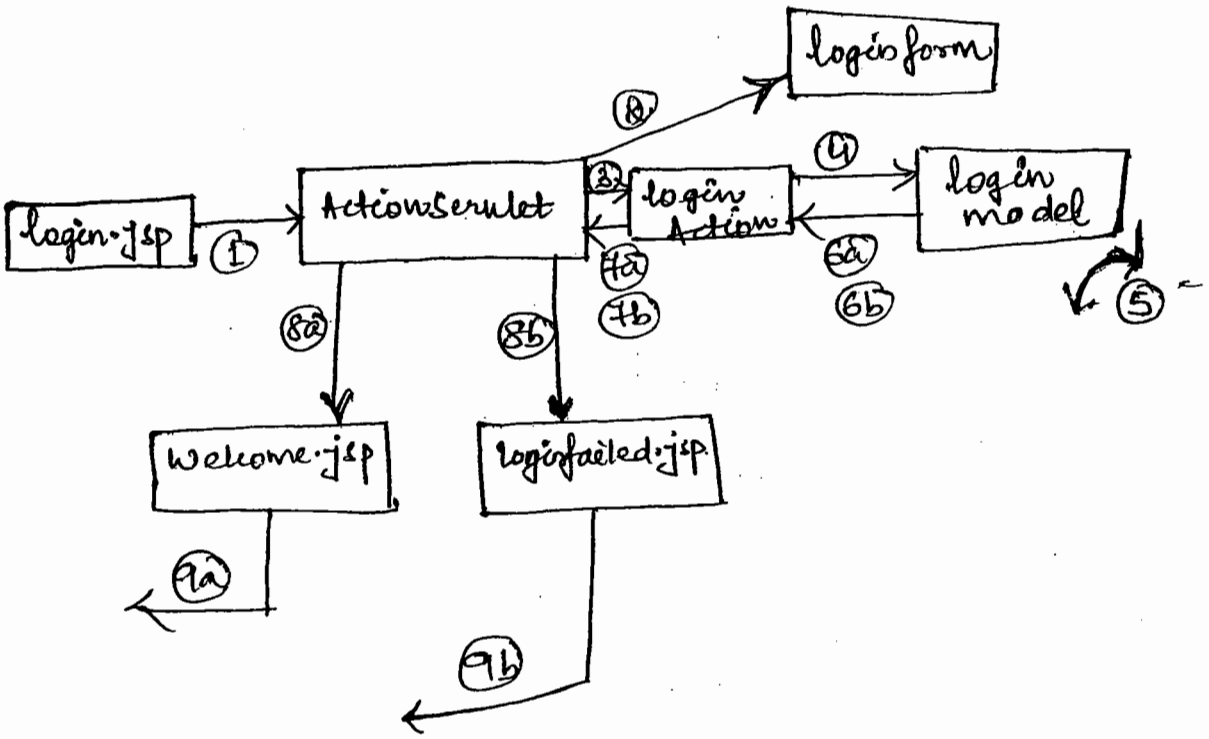
Step5:- develop struts-config.xml.

Step6:- Struts enable the web application i.e web.xml.
In web.xml registering ActionServlet, preinitializing it & making it the front-controller.

Step7:- Configuring the application files.
i.e place xml files into WEB-INF.
Struts jar files into "lib" folder.
form beans and action classes.
class file in "classes" folder.
jsps into root folder (or sub folder of the root folder)

note:- model component developers developed model component related class file also to be placed into "classes" folder.

Q) Develop a struts application to implement the use-case of user authentication?



note:- boolean isAuthenticated(String user, String pwd) is the prototype of model component method.

1:- end-user enters username and pwd into the input page developed by login.jsp and submits the web form. Action (created) Servlet, being the front-controller, receives the client requests and capturing the user-input.

2:- ActionServelet populating formbean fields with user-input.

3:- ActionServelet invoking the execute() method of action instance.

4:- execute() method is invoking isAuthenticated() method of model component.

5:- business method of model component processing the user input i.e verifying the correctness of the username and password against something.

6a:- model component returning "true" to subcontroller component as the user name and password are found to be correct.

6b:- isAuthenticated() method returning "false" to execute() method as it finds that the user name or password is wrong.

7a:- Action returning view mapping information to ActionServlet ie logical name of the "welcome.jsp".

7b:- Subcontroller intimating to the front controller about "loginfailed.jsp" as the target view.

8a & 8b:- ActionServlet switching the control to the response page using forward mechanism of request dispatching.

9a & 9b:- view components producing the appropriate response for the client.

loginapplication
- login.jsp
- welcome.jsp
- loginfailed.jsp
- WEB-INF
  - Struts-config.xml
  - web.xml
  - src
    - loginform.java
    - loginAction.java
    - loginModel.java
  - classes
    - com
      - naveshit
        - form
          - LoginForm.class
        - action

LoginAction.class

model

LoginModel.class

lib

commons-beanutils-1.8.0.jar

commons-chain-1.2.jar

commons-digester-1.8.jar

commons-logging-1.0.4.jar

struts-core-1.3.10.jar ✓

struts-taglib-1.3.10.jar

http://localhost:8081/loginapplication

01/07/13

loginForm.java

```
Package com.nareshit.form;
import org.apache.struts.action.ActionForm;
public class LoginForm extends ActionForm.
{
  private String username;
  private String password;
  public void setUsername (String username)
  {
    this.username = username;
  }
public String getUsername()
  {
    return username;
  }
public void setPassword (String pwd)
  {
```

```
    this.password = pwd;
    }
    public String getPassword()
    {
      return password;
    }
  } // formbean
```

LoginAction.java

```
Package com.nareshit.action;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionForm;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.nareshit.form.LoginForm;
import com.nareshit.model.LoginModel;
public class LoginAction extends Action
{
  LoginModel lm = new LoginModel();
  public ActionForward execute(ActionMapping mapping,
    ActionForm form, HttpServletRequest request,
    HttpServletResponse response)
  {
    String viewinfo = "notok";
```

```
LoginForm lf = (LoginForm) form;
String user = lf.getUsername();
String pwd = lf.getPassword();
boolean flag = ln.isAuthenticated(user, pwd);
if(flag)
 viewinfo = "ok";
ActionForward forward = mapping.findForward
                        (viewinfo);

return forward;
}
}
```

struts-config.xml

```
<struts-config>
 <form-beans>
  <form-bean name = "loginform"
  type = "com.narendra.form.LoginForm" />
 </form-beans>
 <action-mappings>
  <action type = "com.narendra.action.LoginAction" name
      = "loginform" path = "/login">
   <forward name = "ok" path = "/welcome.jsp" />
   <forward name = "notok" path = "/loginfailed.jsp" />
  </action>
 </action-mappings> </struts-config>
```

web.xml

```
<web-app>
  <servlet>
  <servlet-name>frontcontroller</servlet-name>
  <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

  <load-on-startup> 1 </load-on-startup>

  </servlet>

  <servlet-mapping> ~~servlet-name~~
    <servlet-name>frontcontroller</servlet-name>
    <url-pattern> *.do </url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file> login.jsp </welcome-file>

  </welcome-file-list>

</web-app>
```

14/11/13

loginfailed.jsp

```
<HTML>
<BODY BGCOLOR="wheat">
<H1> Login Failed. Invalid username or password </H1>
</BODY>
</HTML>
```

welcome.jsp

```
<HTML>
<BODY BGCOLOR="cyan">
<H1> Welcome to our website </H1>
</BODY>
</HTML>
```

login.jsp

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

<HTML>
  <BODY BGCOLOR="cyan">
  <CENTER>
   <H1> Login Screen </H1>
   <html:form action="/login" method="POST">
   username <html:text property="username" />
   <BR><BR>
   Password <html:password property="password" />
   <BR><BR>
   <html:submit> login </html:submit>
  </html:form>
  </CENTER>
 </BODY>
</HTML>
```

loginModel.jsp

```
package com.nareshit.model
public class LoginModel
{
  public boolean isAuthenticated (String user, String pwd)
  {
    boolean flag = false;
    if (user.equals (pwd))
    flag = true;
    return flag;
  }
}
```

note :- before compiling the action class & form bean, place 2 jarfiles into classpath.

1) servlet-api.jar
2) struts-core-1.3.10.jar.

notepad "struts.bat" tree/f.

for eg :-

```
SET CLASSPATH = .;D:\servlet-api.jar;D:\struts-
                core-1.3.10.jar.
```

Q) what happens in the background when login application deployed into the web container?

⇒ web container reads web.xml as soon as the application is deployed.

⇒ container loads ActionServlet class and instantiates it.

⇒ Container creates ServletConfig object for ActionServlet and invokes its init method by supplying Servlet Config object reference as argument.

⇒ Within the init method of ActionServlet, code is written to read struts-config.xml. i.e as soon as the Struts application is deployed, framework knows the complete generic flow control information of the application.

Q) What happens in the background when login.jsp is executed?

⇒ JSP engine executes login.jsp when the user types url in the browser after application deployment as it is made the home page.

http://localhost:8081/loginapplication

a) appends ".do" extension to "/login", which is the value of action attribute of <form> tag.

b) appends session id to the URL using URL rewriting concept.

c) generates HTML tag i.e <FORM> tag & writes to the browser stream.

⇒ <html:text>, <html:password> & <html:submit> tags write corresponding HTML tags to the browser stream.

⇒ dynamic, pure HTML content generated by login.jsp is sent to the browser via webserver.

⇒ Browser executes that dynamic web content and renders the webform to the end-user.

Q) What happens in the background when user submits the login form by entering username and password?

⇒ Based on the action path (from the input screen), framework identifies appropriate action class and form bean.

⇒ form bean is instantiated, populated with user input. LoginAction instance created. LoginAction instance created. Its execute() method is called by supplying loginForm instance as argument.

⇒ Based on the loginModel's response execute() method returns ActionForward object that holds views mapping information to framework.

⇒ ActionForward is unwrapped and by using the logical name of the jsp path, control is switched to the appropriate jsp.

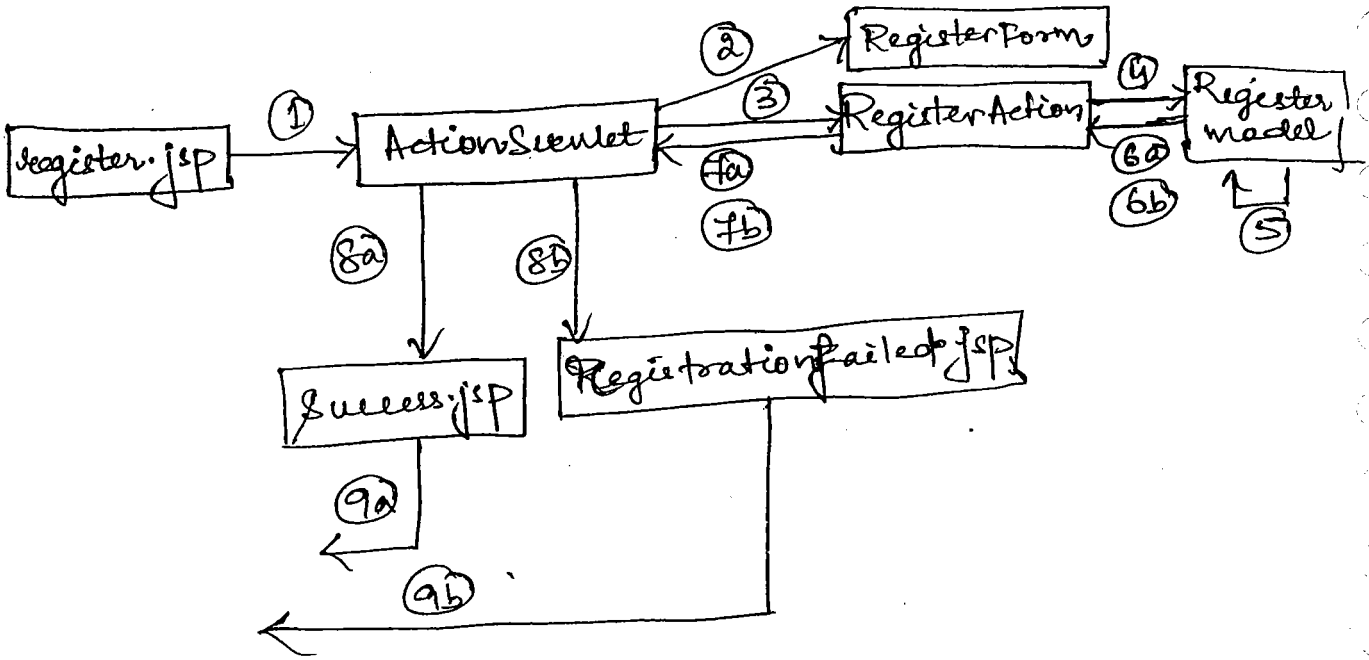
⇒ jsp produces the response and via web server the response page is sent to the client.

12/11/13

Q) Develop a Struts application that implements the use case of user-registration?

Ans) REGISTRATION (username, password, emailid) is the database table.

⇒ boolean registerUser(String user, String pwd, String email) is the prototype of the model method.



registrationapplication
- register.jsp
- Success.jsp
- Registrationfailed.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - src
    - RegisterForm.java
    - RegisterAction.java
    - RegisterModel.java
  - classes
    - *.class
  - lib
    - *.jar (.6 jar file)
    - driver jar file

http://localhost:8081/registrationapplication

Web.xml

⇒ make register.jsp the homepage.

struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="registerform" type="com.nareshit
        .form.RegisterForm" />
  </form-beans>
  <action-mappings>
    <action path="/register" name="registerform"
    type="com.nareshit.action.RegisterAction">
    <forward name="ok" path="/success.jsp" />
    <forward name="notok" path="/registrationfailed.
                                jsp" />

  </action>
  </action-mappings>
</struts-config>
```

RegisterForm.java

```
package com.nareshit.form;
import org.apache.struts.action.ActionForm;
public class RegisterForm extends ActionForm
  {
    private String username;
    private String password;
    private String emailid; // bean fields.
    // 3 setters & getters
  }
```

RegisterAction.java

```
package com.naresh.action;
import com.naresh.form.RegisterForm;
import com.naresh.model.RegisterModel;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class RegisterAction extends Action
{
RegisterModel rm = new RegisterModel();
public ActionForward execute(ActionMapping
mapping, ActionForm form, HttpServletRequest request,
HttpServletResponse response).
{
RegisterForm ref = (RegisterForm) form;
String user = ref.getUsername();
String pwd = ref.getPassword();
String emailid = ref.getEmailid();
boolean flag = rm.registerUser(user, pwd,
                               emailid);
if(flag)
return mapping.findForward("ok");
else
return mapping.findForward("notok");
}
}
```

register.jsp

```
<%@ taglib prefix="html"
  uri="http://stauts.apache.org/tags-html" %>
<HTML>
  <BODY BGCOLOR="cyan">
  <CENTER>
    <H1> Registration Screen </H1>
    <html:form action="/register" method=
      "POST"> username<html:tent property="userna
          me" />
      <BR><BR>
    password<html:password property="password"/>
    <BR><BR>
    emailid<html:tent property="emailid"/>
    <BR><BR>
    <html:submit>register</html:submit>
  </html:form>
  </CENTER>
  </BODY>
</HTML>
```

success.jsp

```
<HTML>
<BODY BGCOLOR = "cyan">
<H1> Registered Successfully </H1>
</BODY>
</HTML>
```

ojdbc14.jar
ojdbc6.jar.

registrationfailed.jsp

```
<HTML>
<BODY BGCOLOR = "wheat">
<H1> Registration failed. Please try after sometime.
</H1>
</BODY>
</HTML>
```

javac -d . *.java

13/11/13

RegisterModel.java

```
package com.naresh.model;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PrepareStatement;
import java.sql.SQLException;
public class RegisterModel
{
  public boolean RegisterUser(String user, String pwd,
                              String mailid)
  {
```

```
boolean flag = false;
Connection con = null;
PreparedStatement ps = null;
try
{
Class.forName("oracle.jdbc.driver.OracleDriver");
con = DriverManager.getConnection("jdbc:oracle:thin:@
 localhost:1521:server","scott","tiger");
ps = con.prepareStatement("INSERT INTO REGISTRATION
     VALUES(?,?,?)");
 ps.setString(1, user);
 ps.setString(2, pwd);
 ps.setString(3, mailid);
 ps.executeUpdate();
 flag = true;
 }
Catch(ClassNotFoundException e)
 {
   e.printStackTrace();
 }
Catch(SQLException e)
 {
   e.printStackTrace();
 }
 finally
```

```
try
{
  if(ps != null)
    ps.close();
  if(con != null)
    con.close();
}
catch(SQLException e)
{
  e.printStackTrace();
}
}
return flag;
} // business method of the model component.
}
```

Q) Modify loginModel so as to verify user authentication details against the database.

```
package com.nanehit.model;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class loginModel
{
  public boolean isAuthenticated(String user, String
                                 pwd)
  {
```

```
boolean flag = false;
Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;
try
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Con = DM.getConnection("jdbc:oracle:thin:@localhost:1521:
 server","scott","tiger");
ps = con.prepareStatement("SELECT * FROM REGISTRATION
 WHERE USERNAME = ? AND PASSWORD = ?");
ps.setString(1, user);
ps.setString(2, pwd);
rs = ps.executeQuery();
flag = rs.next();
}
Catch (ClassNotFoundException e)
{
e.printStackTrace();
}
Catch (SQLException e)
{
e.printStackTrace();
}
finally
{
try
{
```

```
if(rs != null)
   res.close();
if (ps!=null)
   ps.close();
if (con!=null)
   con.close();
}
Catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
  return flag;
  } // business method of model component
}
```

Q) How to centralise database connection creation and resources releasing code in an application?

⇒ Develop a utility class.

```
package com.naresh.model;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class DBUtil
   {
```

```
Static
{
  try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
   }
  Catch (ClassNotFoundException e)
    {
      e.printStackTrace();
    }
  } // static initializer
public static Connection getConnection() throws SQLException
  {
   return DriverManager.getConnection("jdbc:oracle:thin:
   @localhost:1521:server", "scott", "tiger");
}
public static void close (ResultSet rs, Statement st,
                          Connection con)
    {
      try
       {  if(rs!=null)
            rs.close();
          if (st!=null)
            st.close();
          if (con!=null)
            con.close();
       }
      catch (SQLException e)
```

```
        {
          e.printStackTrace();
        }
      }
    }
  } // DBUtil
```

4/11/13

Q) How to create a hyperlink in a jsp of Struts application?

⇒ One view shouldn't decide the ~~target~~ page of the hyperlink. It becomes the gross violation of Mvc. Only controller should ~~target~~ decide the target view.

⇒ In logical name based manner hyperlink has to be created in a jsp targeting other page.

⇒ Three steps are involved in the creation of a hyperlink in a jsp.

Step1:- configure a builtin action class.
org.apache.struts.actions.ForwardAction in struts configuration file once for each link.

for eg:-

```
<action path="/registerpage" type="org.apache.struts.actions.
ForwardAction" parameter="/register.jsp" />
```

Step2:- create the link in a jsp by using <html:link> tag.

for eg:-

```
<html:link="/registerpage">Click here</html:link>
```

Step3:- place an additional jar file in lib folder of the application (directory structure). struts-extras-1.3.10.jar.

Q) Develop a struts application in which, registration and login use-cases are implemented?

Ans)

registerloginapplication
- index.jsp
- register.jsp
- login.jsp
- registrationfailed.jsp
- registrationsuccess.jsp
- loginfailed.jsp
- loginsuccess.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - src
    - LoginForm.java
    - LoginAction.java
    - LoginModel.java
    - RegisterForm.java
    - RegisterAction.java
    - RegisterModel.java
    - DBUtil.java
  - classes
    - db.properties
    - *.class
  - lib
    - *.jar(6 jar files)
    - *.jar(Driver jar file)
    - struts-extras-1.3.10.jar

http://localhost:8081/registerloginapplication

db.properties

driver = ~~oracle.jdbc.driver~~ oracle.jdbc.driver.OracleDriver

cs = jdbc:oracle:thin:@localhost:1521:xe

username = scott

Password = tiger

index.jsp

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

<HTML>
   <BODY BGCOLOR = "cyan">
     <CENTER>
<H1><html:link action="/loginpage"> login here </html:link></H1>
<H1><html:link action = "/registerpage"> register here </html:link>
                                                        </H1>
      </CENTER>
      </BODY>
      </HTML>
```

struts-config.xml

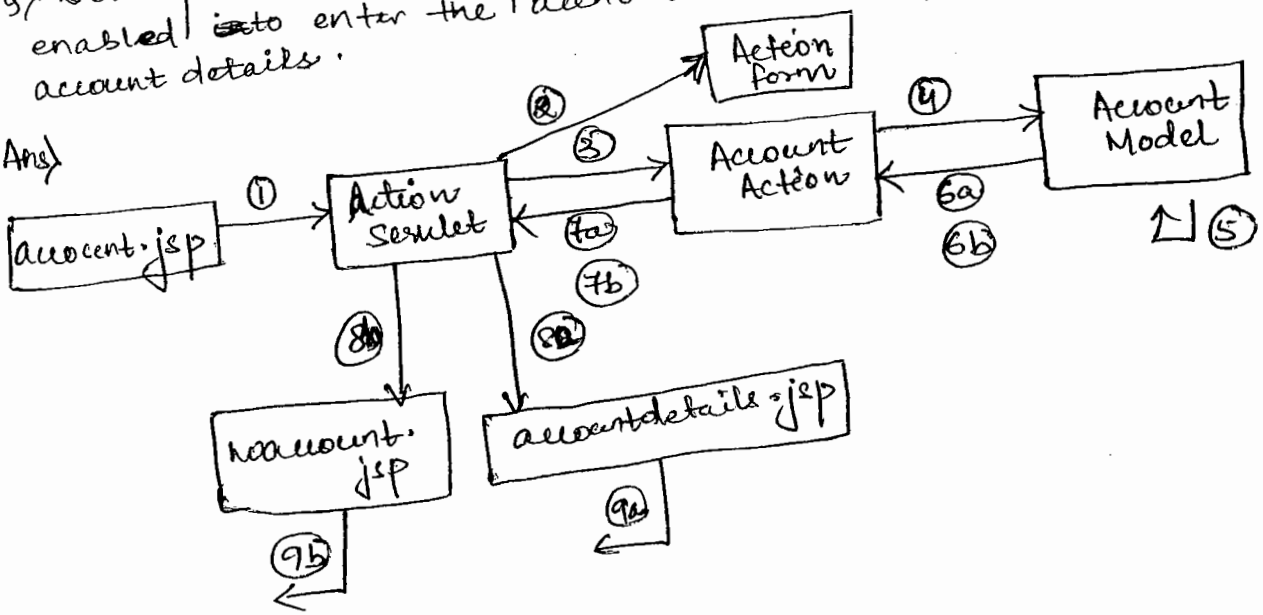
```
<struts-config>
<form-beans>
    <form-bean name = "loginform" type = "com.nareshitform.
    LoginForm" />
    <form-bean name = "registerform" type = "com.nareshit.
    form.RegisterForm" />

</form-beans>
<action-mappings>
    <action name = "registerform" path = "/register"
    type = "com.nareshit.action.RegisterAction" >
    <forward name = "ok"
      path = "/registerationsuccess.jsp" />
    <forward name = "notok"
      path = "/registrationfailed.jsp" />

    </action>
    <action name = "loginform" path = "/login"
      type = "com.nareshit.action.LoginAction" >
```

```
<forward name="ok" path="/loginsuccess.jsp" />
<forward name="notok" path="/loginfailed.jsp" />
</action>
<action path="/loginpage"
 type="org.apache.struts.actions.ForwardAction"
  parameter="/login.jsp" />
<action path="/registerpage"
 type="org.apache.struts.actions.ForwardAction"
 parameter="/register.jsp" />
</action-mappings>
</struts-config>
```

15/11/12

8) Develop a struts application in which end-user should be enabled into enter the accno to the webpage and get the account details.

Ans)



(4) execute() method calling getAccountDetails() method of the model component.

(6a) → model component method returning account object to the execute method as record is found in the database.

(6b) → model component returning null to sub-controller as record is not found in the database.

(7a) → execute() method returning success view mapping information to the framework after storing account object into the request scope.

```
accountapplication
    account.jsp
    accountdetails.jsp
    noaccount.jsp
    WEB-INF
        web.xml
        struts-config.xml
        src
        AccountForm.java
        AccountAction.java
        AccountModel.java
        Account.java (DTO)
        DBUtil.java
        classes
            *.class
                db.properties
        lib
            *.jar (6+1 jar files)
```

http://localhost:8081/accountapplication

web.xml

make account.jsp the home page.

struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="accountform" type="com.nareshit.form.
                                        ActionForm" />

  </form-beans>
  <action-mappings>
    <action path="/getaccount" type="com.nareshit.action.
        AccountAction" name="accountform">
      <forward name="success" path="/accountdetails.jsp" />
      <forward name="failure" path="/noaccount.jsp" />

    </action>
  </action-mappings>
  <message-resources parameter="ApplicationResources" />

</struts-config>
```

AccountForm.java

```
package com.nareshit.form;
import org.apache.struts.action.ActionForm;
public class AccountForm extends ActionForm
 {
    private int accno;
    public void setAccno(int accno)
      { this.accno=accno;
    }
     public int getAccno()
       { return accno;
    } }
```

Account.java

```
package com.nareshit.model
private class Account
 {
   private int accno;
   private String name;
   private float balance;
 } // 3 setters & 3 getters
```

AccountModel.java

```
package com.nareshit.model;
import java.sql.*;
public class AccountModel
 {
   public Account getAccountDetails(int accno)
    {
     Account acc = null;
     Connection con = null;
     PreparedStatement ps = null;
     ResultSet res = null;
   try {
        con = DBUtil.getConnection()
        ps = con.prepareStatement("SELECT * FROM
         ACCOUNT WHERE ACCNO = ?");
       ps.setInt(1, accno);
       res = ps.executeQuery();
       if (rs.next())
          {
```

```
        acc = new Account();
        acc.setAccno(rs.getInt(1));
        acc.setName(rs.getString(2));
        acc.setBalance(rs.getFloat(3));
      }
    } // try
  catch (SQLException e)
    {
      e.printStackTrace();
    }
    finally
    {
      DBUtil.close(rs, ps, con);
    }
    return acc;
  } // business method.
}
```

AccountAction.java

```
package com.naresh.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import com.naresh.actionsst.form.AccountForm;
import com.naresh.model.AccountModel;
import com.naresh.model.Account;
public class AccountAction extends Action
  {
```

```
AccountModel am = new AccountModel();
public ActionForward execute (ActionMapping mapping, Action
    form form, HttpServletRequest request, HttpServletResponse
                                                response)
  {
    AccountForm af = (AccountForm) form;
    int ano = af.getAcno();
    Account acc = am.getAccountDetails (ano);
    if (acc != null)
    {
      request.setAttribute ("account", acc); // to make
                    it available to view.
      return mapping.findForward ("success");
    }
    else
      return mapping.findForward ("failure");
  }
}
```

Account.jsp

```
<%@ taglib prefix = "html"
    uri = "http://struts.apache.org/tags-html" %>
<HTML>
    <BODY BGCOLOR = "cyan">
    <CENTER>
        <H1> Account Details Retrieval </H1>
```

```
<html:form action = "/getaccount">
Aceno <html:text property = "aceno" /><BR><BR>
<html:submit> get account details </html:submit>
</html:form>
</CENTER>
</BODY>
</HTML>
```

noaccount.jsp

```
<HTML>
  <BODY BGCOLOR = "wheat">
    <H2> with that number no A/c exists </H2>
  </BODY>
</HTML>
```

accountdetails.jsp

```
<%@ taglib prefix = "bean"
    uri = "http://struts.apache.org/tags-bean" %>
<HTML>
  <BODY BGCOLOR = "cyan" >
  Aceno : <bean:write name = "account" property = "aceno" />
  <BR>
    A/c holder name: <bean:write name = "account"
      property = "name" /><BR>
  Balance Rs. <bean:write name = "account"
      property = "balance" />
</BODY>
</HTML>
```

18/11/13

Q) How to prevent hardcoding of Database connection details in DBUtil class?

⇒ read database connection details specified in properties file into java.util.Properties object.

⇒ from properties object, retrieve database connection details

DBUtil.java

```
package com.naveshit.model;
import java.sql.*;
import java.util.properties;
public class DBUtil
{
  private static Properties p = new Properties();
  static
  {
    try {
      p.load(DBUtil.class.getClassLoader().getResourceAs.
        Stream("db.properties"));
    Class.forName(p.getProperty("driver"));
  }
  catch(Exception e)
    {
      e.printStackTrace();
    }
}// static initializer
public static Connection getConnection() throws.
    SQLException
    {
      return
        DriverManager.getConnection(p.getProperty("cs"),
```

```
p.getProperty("username"), p.getProperty("password"));
    }
    ... //close method
}// DBUtil
```

Q) Develop a struts application that enables the end-user to enter balance range in to a text box and displays the accountdetails that fall under that range?

⇒ note:- file names and flow diagram from the previous application (accountapplication).

accountsapplication
- account.jsp
- noaccount.jsp
- accountdetails.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - src
    - Account.java
    - AccountModel.java
    - AccountAction.java
  - classes
    - db.properties
    - *.class
  - lib
    - *.jar (6+1)

http://localhost:8081/accountsapplication

4:- execute method is calling business method of AccountModel which has the following prototype.

public list<Account> getAccountsDetails (float lower, float upper).

6a:- model returning Arraylist object to controller.
6b:- model returning null to controller.

note :- Account.java, web.xml, struts-config.xml, DBUtil.java & db.properties from the previous application.

account.jsp

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

<HTML>
  <BODY BGCOLOR = "cyan">
    <CENTER>
      <H1> Accounts Details Retrieval </H1>
      <html:form action = "/getaccount">
      BALANCE RANGE (lower-upper) <html:text
        property = "balrange" /> <BR> <BR>
      <html:submit> get accounts details </html:
        Submit>
      </html:form>
    </CENTER>
  </BODY>
</HTML>
```

AccountForm.java

```
package com.nareshit.form;*
import org.apache.struts.action.ActionForm;
Public class AccountForm entends ActionForm
{
  private String balrange;
  public void setBalrange(string balrange)
  {
    this.balrange = balrange;
  }
  public String getBalrange()
  {
    return balrange;
  }
}
```

AccountAction.java

```
package com.nareshit.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import com.nareshit.form.Accountform;
import com.nareshit.model.AccountModel;
import com.nareshit.model.Account;
import java.util.List;
import java.util.StringTokenizer;
public class AccountAction entends Action.
{
  AccountModel am = new AccountModel();
  public ActionForward execute(
  )
```

```
{
  AccountForm af = (AccountForm) form;
  String br = af.getBalrange();
  StringTokenizer st = new StringTokenizer(br,"-");
  float lower = Float.parseFloat(st.nextToken());
  float upper = Float.parseFloat(st.nextToken());
  List<Account> accounts = am.getAccountsDetails
                           (lower, upper);

    if (accounts != null)
     {
      request.setAttribute("accounts", accounts);
      return mapping.findForward("success");
     }
    else
      return mapping.findForward("failure");
  }
}
```

19/11/13

AccountModel.java

```
package com.narushet.model;
import java.sql.*;
import java.util.List.*;
import java.util.ArrayList.*;
public class AccountModel
{
  public List<Account> getAccountsDetails(float lower, float
                                          upper)
  {
```

```
List<Account> accounts = null;
Connection con = null;
PreparedStatement ps = null;
ResultSet rs = null;
try
{
 con = DBUtil.getConnection();
 ps = con.prepareStatement("SELECT * FROM ACCOUNT WHERE
  BALANCE BETWEEN ? AND ?");
 ps.setFloat(1, lower);
 ps.setFloat(2, upper);
 rs = ps.executeQuery();
if(rs.next()).
{
 accounts = new Arraylist<Account>();
do
{
  Account acc = new Account();
  acc.setAccno(rs.getInt(1));
  acc.setName(rs.getString(2));
  acc.setBal(rs.getFloat(3));
  accounts.add(acc);
 } while(rs.next());
} // if
}// try
catch(SQLException e)
 { e.printStackTrace();
 }
```

```
finally
  {
   DBUtil.close(rs, ps, con);
  }
  return accounts;
 } // business method.
}
```

AccountDetails.jsp

```
<%@ taglib prefix="bean"
    uri="http://struts.apache.org/tags-bean" %>
<%@ taglib prefix="logic"
    uri="http://struts.apache.org/tags-logic" %>

<HTML>
  <BODY BGCOLOR="cyan">
    <CENTER>
     <H1>Account Details</H1>
     <TABLE BORDER="1">
       <TR>
         <TH>ACCNO</TH>
         <TH>NAME</TH>
         <TH>BALANCE</TH>
       </TR>
      <logic:iterate name="accounts" scope="request"
        id="account">
      <TR>
        <TD><bean:write name="account"
          property="accno" /></TD>
        <TD><bean:write name="account"
          property="name" /></TD>
        <TD><bean:write name="account"
          property="balance" /></TD>
      </TR>
```

```
</logic:iterate>
</TABLE>
</CENTER>
</BODY>
</HTML>
```

no account.jsp

```
<HTML>
 <BODY BGCOLOR="wheat">
  <H1> Within that balance Range no accounts exist </H1>
 </BODY>
</HTML>
```

Q) What happens in the background when find-forward() method is called on ActionMapping object in the execute() method of action class?

⇒ for each <forward> tag of the use-case, one ActionForward object is created.

⇒ all the ActionForward objects are stored in Map object.

⇒ Map object is wrapped into ActionMapping object.

⇒ now, framework is invoking execute() method of action class by supplying ActionMapping as one of the arguments.

⇒ when findForward() method is called on the mapping object by supplying logical name of the view path as argument, ActionForward object is not created. Instead search is made in the map and reference of already existing ActionForward object is returned.

⇒ one <forward> is a URL alias of a view component.

⇒ object oriented representation of one <forward> tag info is nothing but ActionForward object.

⇒ framework resolves the view path based on the logical name wrapped in ActionForward object.

Q) How to implement exception handling in a Struts application?

⇒ handling of exceptions in model component doesn't come under exception handling in Struts application. i.e. presentation layer.

⇒ In view components no need of handling the exceptions.

⇒ When controller component is communicating with the business method of model component there is a chance of exception raising. Dealing with such exceptions is nothing but exception handling in a struts application.

⇒ dealing with exception in the action class while it is communicating with model component is nothing but exception handling in a struts application.

⇒ unless exceptions are propagated by model component to action class, exception handling can't be implemented in a Struts Application.

⇒ If exception handling is not implemented in presentation layer, wrong information goes to the end-user.

⇒ Handling exceptions in a struts application is of 2 types.

1) programmatical exception handling.
2) declarative exception handling.

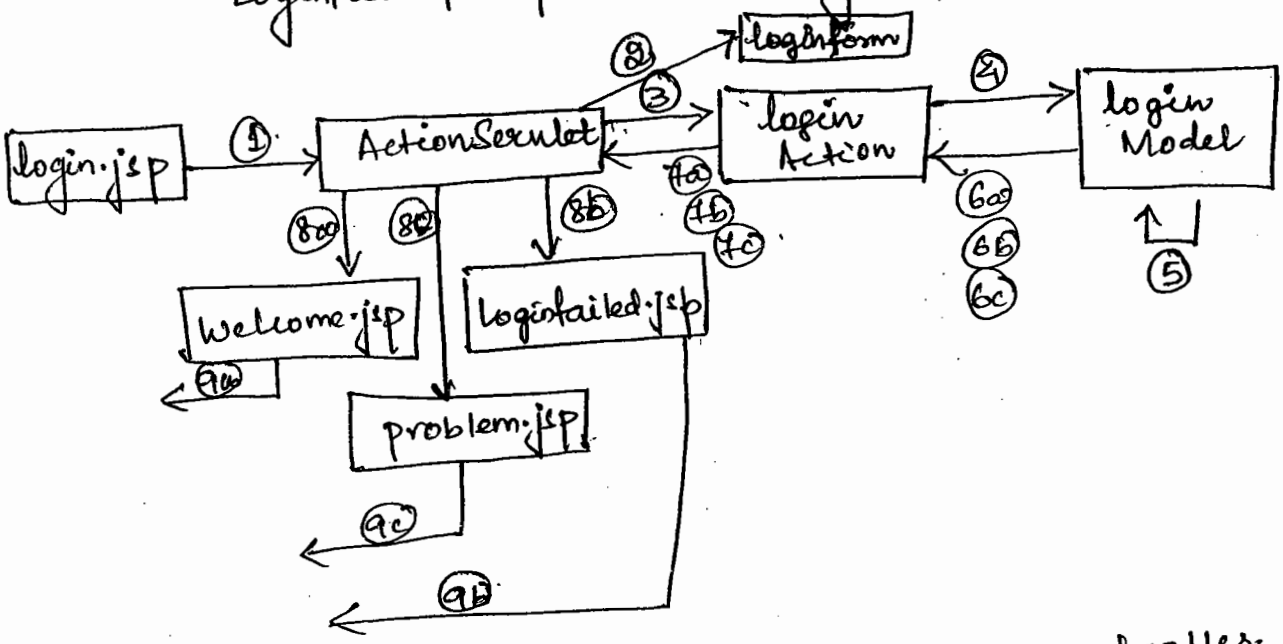
20/11/13

Q) Implement programmatical validation in "Loginapplication".

⇒ within the action class, Struts developer placing the business method call in try block and implementing corresponding catch blocks is nothing but programmatical exceptional handling.

⇒ login Model's method should have the following prototype & that method should propogate the specified exceptions to the controller. Then only, exception handling can be implemented in action class.

public void isAuthenticated (String user, String pwd) throws LoginFailedException, ProcessingException.



(6a) :- model method returning nothing to the controller but control.

(6b) :- model method propogating loginfailed exception to the controller component. (during zero record on table it found).

(6c) :- model method propogating processingException to the controller as it encountered abnormal event while communicating with the database.

progeneploginapplication
- login.jsp
- loginfailed.jsp
- welcome.jsp
- problem.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - src
    - LoginModel.java
    - LoginAction.java
    - LoginForm.java
    - LoginFailedException.java
    - ProcessingException.java
    - DBUtil.java
  - classes
    - *.class
    - db.properties

http://localhost:8081/progeneploginapplication

problem.jsp

```
<HTML>
  <BODY BGCOLOR = "cyan">
    <H1> Server problem. please try to login after sometime </H1>
  </BODY>
</HTML>
```

loginFailedException.java

```
package com.nareshit.exception;
public class LoginFailedException extends Exception
{
}
```

ProcessingException.java

```
package com.nareshit.exception;
public class ProcessingException extends Exception
{
}
```

loginModel.java

```
package com.nareshit.model;
import java.sql.*;
import com.nareshit.exception.LoginFailedException;
import com.nareshit.exception.ProcessingException;
public class LoginModel
{
public void isAuthentical (String user, String pwd)
    throws LoginFailedException, ProcessingException.
  {
    Connection con=null;
    PreparedStatement ps = null;
    ResultSet rs= null;
  try
  {
```

```
con = DBUtil.getConnection();
ps=con.prepareStatement("SELECT * FROM REGISTRATION
        WHERE USERNAME = ? AND PASSWORD = ?");
    ps.setString (1, user);
    ps.setString (2, pwd);
  rs = ps.executeQuery();
if (!rs.next())
    throw new LoginFailedException();// throwing
  }
catch (SQLException e)
  {
    e.printStackTrace();
    throw new ProcessingException(); // rethrowing
  }
  finally
    { DBUtil.close (res, ps, con);
    }
  } // business method of the model component.
}
```

LoginAction.java

```
public class LoginAction extends Action
  {
    LoginModel lm = new LoginModel();
    public ActionForward execute (...)
      {
        String viewinfo = "ok";
        LoginForm lf=(LoginForm) form;
        String user = lf.getUsername();
        String pwd = lf.getPassword();
        try
```

```
{
  lm.isAuthenticated(user, pwd);
}
catch(LoginFailedException e)
{
  viewInfo = "notok";
}
catch (ProcessingException e)
{
  viewInfo = "abnormal";
}
ActionForward forward = mapping.findForward(viewInfo);
return forward;
}
}
```

note:- In struts configuration file add third <forward> within <action> tag

```
<forward name="abnormal" path="/problem.jsp" />
```

2) Modify "accountapplication" so as to implement programmatical exception handling.

```
public Account getAccountDetails(int accno) throws AccountNotFoundException, ProcessingException
```

AccountNotFoundException.java

```
package com.naresh.exception;
public class AccountNotFoundException extends Exception
{
  public AccountNotFoundException(String msg)
  {
    super(msg);
  }
}
```

AccountAction.java

```
public class AccountAction extends Action.
{ AccountModel am = new AccountModel();
  public ActionForward execute(...)
  {
   String viewmapping info = "success";
   AccountForm af = (AccountForm) form;
   int accno = af.getAccno();
  try
   {
    Account acc = am.getAccountDetails(accno);
    request.setAttribute("account", acc);
  }
  catch (AccountNotFoundException e)
   {
    S.O.P(e.getMessage());
   viewmappinginfo = "failure";
   }
  catch (ProcessingException e)
   {
    S.O.P(e);
    viewmappinginfo = "problem";
   }
   return mapping.findForward(viewmappinginfo);
  } //execute.
}
```

note:- in struts-config.xml file add
third <forward> within <action> tag.

<forward name = "problem" path = "/problem.jsp" />

AccountDaoImpl.java

```
public Account getAccountDetails (int accno) throws AccountNot
        FoundException, ProcessingException
    {
      ......
      try
        {.....
          if(rs.next())
            {
              ....
            }
          else
            throw new AccountNotFoundException(accnot
                  "doesn't exist in database");
      } //try
    catch (SQLException e)
      {
        throw new ProcessingException();
      }
      finally
        {
          {....
          }
      return acc;
    } //execute
}
```

22/11/13

Q) What is a resource bundle in the content of a Structs application ?

⇒ Resource Bundle is a properties file in a struts application whose standard name is "ApplicationResources.properties".

⇒ In a resource bundle appropriate key value pairs of text is placed by the application developer.

⇒ At runtime struts read the content of the resource bundle

⇒ Struts uses resource bundle as information repository whose content it uses to dynamically populate jsps.

⇒ Resource Bundle is used in declarative exception handling, declarative validations and i18n.

⇒ Using a resource bundle in a Struts application involves the following steps.

Step1 :- develop a text file with name (key), value pairs according to application requirement and store it into "classes" folder with the file name "ApplicationResources.properties".

Step2 :- register the resource bundle with struts making the following entry in struts-config.xml.

<message-resources parameter = "ApplicationResources" />

Q) Explain about declarative Exception Handling ?

⇒ Instead of developer Implementing the exception handlers in the action class, declaring the exception handler requirements to struts through configuration file and framework providing the exception handlers for the action classes is nothing but declarative exception handling.

⇒ Implementing declarative exception handling in a struts application involves the following steps.

Step1 :- develop a resource bundle with appropriate error keys and corresponding messages.

Step2 :- make use of <exception> tags within the <action> tag to declare the required exception handlers for the action class.

Step3:- Specify exception specification with java.lang.Exception to the execute() method of action class. Dont implement any exception handling in execute() method.

Step4:- make use of <html:errors/> tag in the jsp that is designed to display exception message to the user.

Q modify the previous application (pregencpaccountapplication) so as to implement exception handling declaratively?

declencpaccountapplication
- account.jsp
- accountdetails.jsp
- exception.jsp (any name)
- WEB-INF
  - *.xml
  - src
    - *.java
  - classes
    - db.properties
    - ApplicationResources.properties
      - *.class
  - lib
    - (6+1)*.jar

http://localhost:8084/declencpaccountapplication

note:- model related files including AccountNotFoundException and processingException are as they are from the previous application.

⇒ AccountForm, account.jsp, accountdetails.jsp and web.xml also from the previous application.

ApplicationResources.properties

db.search.failed = Account doesn't exist.
db.operation.failed = Server problem. Please try later.

struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="accountform"
    type="com.narasit.form.AccountForm" />
  </form-beans>
  <action-mappings>
    <action name="accountform"
    type="com.narasit.action.AccountAction" path=
                                    "/getaccount">

      <exception
        type="com.narasit.exception.AccountNotFoundException"
        path="/exception.jsp" key="db.search.failed" />

      <exception
        type="com.narasit.exception.ProcessingException"
        path="/exception.jsp" key="db.operation.failed" />

      <forward name="success" path="/accountdetails.
                                    jsp" />

    </action>
  </action-mappings>
  <message-resources
    parameter="ApplicationResources" />
</struts-config>
```

AccountAction.java

```
// import all except the exceptions
public class AccountAction extends Action
{
  AccountModel am = new AccountModel();
  public ActionForward execute (ActionMapping mapping,
  ActionForm form, HttpServletRequest request, HttpServlet
    Response response) throws Exception
  {
    AccountForm af = (AccountForm) form;
    int accno = af.getAccno();
    Account acc = am.getAccountDetails (accno);
  request.setAttribute ("account", acc);
  return mapping.findForward ("success");
  }
}
```

exception.jsp

```
<%@ taglib prefix = "html"
  uri = "http://struts.apache.org/tags-html" %>
  <HTML>
    <BODY BGCOLOR = "wheat">
    <FONT COLOR = "red" SIZE = "6">
      <html:errors/>
    </FONT>
  </BODY>
</HTML>
```

22/11/13

Q) How does declarative exception handling work in the backgr

⇒ While execute() method is invoking the business method of the model component, if the business method propogates the exception execute method, framework provided exception handler comes in action.

⇒ Framework provided exception handler does the following things.

1) creates org.apache.struts.action.ActionMessage object encapsulating resource bundle key in it.

2) places ActionMessage object in scope.

3) indicates to the front-controller about the jsp to which it has to switch the control.

⇒ <html:errors> tag does the following things.

1) retrieves ActionMessage object from scope and gets the resource bundle key from it.

2) Searches in the resource bundle for the corresponding message by using the key as search criteria.

3) writes the message retrieved from the resource bundle to the browser stream.

Q) How are form beans classified?

⇒ form beans are of two types.

① Static form beans

② dynamic form beans.

⇒ Application developer developed form beans are known as static form beans.

⇒ Framework provided form beans are known as dynamic form beans.

⇒ using a dynamic form bean in a struts application involves 3 steps.

Step 1 :- configure org.apache.struts.action.DynamicActionForm as type of bean in <form-beans> section in struts configuration file.

Step 2 :- for each input field of the web form, make one <form-property> entry.

Step3:- In action class, retriene form bean data by calling getC) methods using DynaActionForm reference.

Q) Modify first struts application (loginapplication) so as to use dynamic form bean in the application.

Ans)

dynaformloginapplication
- *.jsp
- WEB-INF
  - *.xml
- src
  - LoginAction.java
  - LoginModel.java
- classes
  - *.class
- lib
  - *.jar(6)

http://localhost:8081/dynaformloginapplication

struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="loginform"
      type="org.apache.struts.action.DynaActionForm">
      <form-property name="username"
        type="java.lang.String" />
      <form-property name="password"
        type="java.lang.String" />
  </form-beans>
  <action-mappings>
    ----
  </action-mappings>
</struts-config>
```

loginAction.java

```
.....
import org.apache.struts.action.DynaActionForm;
public class LoginAction extends Action
{
  public ActionForward execute(...)
  {
    DynaActionForm daf=(DynaActionForm) form;
    String user = (String) daf.get("username");
    String pwd =(String) daf.get("password");
    boolean flag = lm.isAuthenticated(user, pwd);
    if(flag)
      returnmapping.findForward("ok");
    else
      returnmapping.findForward("notok");
  }
}
```

Modify "accountapplication" so as to use dynamic formbean.

Struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="accountform"
      type="org.apache.struts.action.DynaActionForm">
      <form-property name="accno"
        type="java.lang.Integer"/>
    </form-bean>
  </form-beans>
</struts-config>
```

AccountAction.java

```
execute()
{
DynaActionForm daf=(DynaActionForm)form;
int accno=(Integer)daf.get("accno");
....
}
```

Q) What are global forwards?

⇒ We have 2 kinds of forwards.

1) action specific forwards.
2) global forwards.

⇒ If a forward is configured within an action tag. (as subtag), it is known as action specific forward and it is available to that action only i.e it is not available to other actions of the application.

⇒ Whenever we have same response from multiple use-cases, instead of configuring same <forward> for action, we can go for global forwards.

for eg.

```
<struts-config>
  <form-beans>
  ....
  </form-beans>
  <global-forwards>
    <forward name="ok" path="/success.jsp" />
    <forward name="notok" path="/failure.jsp" />
  </global-forwards>
  <action-mappings>
    <action...>
    </action>
```

```
<action...>
</action>
</action-mappings>
</struts-config>
```

note :- now 2 same forwards are available to both the actions without the need of configuring them in <action>

25/11/13

Q) what are global exceptions ?

Ans) During declarative exception handling an exception handler configured using <exception> as a sub tag of <action> is nothing but action specific exception handler configuration.

⇒ Such handler is not available to another action class. even though same exception is propagated from the model component.

⇒ Whenever same exception handler is required to be configured to multiple action classes, we go for global exception configuration to reduce the amount of xml configuration in struts configuration file.

forex :-

```
<global-exceptions>
    <exception key = "db.operation.failed"
        type = "com.naresh.exception.ProcessingException"
        path = "/exception.jsp" />
</global-exceptions>
```

VALIDATION IN STRUTS APPLICATIONS

Q) Explain about validation in a Struts Application?

⇒ verification of user input for its completeness and correctness is nothing but performing validation in a web application.

⇒ validation is of two types.

1) client side validation
2) server side validation.

⇒ In a Java web application, JavaScript code that is executed by the browser for validating the user input is nothing but client side validation code.

⇒ for the same web form data, server side validation is performed for 2 reasons.

1) client side validation code might be skipped without execution with compatibility issue or JavaScript disabled in the browser.
2) for security reasons some validation logic can't be placed at client side.

note:- for every client side validation, serverside validation is also performed. Some additional validations also may be performed at serverside.

⇒ Validating the user input stored in form bean is nothing but performing validation in a struts application.

⇒ We have 2 kinds of validation in a struts application.

⇒ programmatical validation

⇒ declarative validation.

⇒ If developer is responsible to implement validation logic in the form beans, it is nothing but programmatical validation.

⇒ If framework providing the validation logic and developer is responsible just to declare the validation requirements in some format, is known as declarative validation.

note:- Almost always declarative validation only is used in real struts applications.

Q) How to implement programmatical validation in a Struts application?

Step1:- Develop a resource bundle with appropriate error keys and corresponding messages.

Step2:- implement validate() method in the form bean & place the appropriate validation logic in that method.

Step3:- enable validation and specify the page to which control has to be switched in case of validation failure.

⇒ validate and input attributes of <action> tag are used for this purpose.

Step4:- In the input page make use of <html:errors/> tag.

Q) Develop a ~~java web~~ struts applications in which, programmatical validation is implemented?

⇒

```
prgvalidationloginapplication
    login.jsp
    welcome.jsp
    logininfailed.jsp
    WEB-INF
        web.xml
        struts-config.xml
    src
        LoginForm.java
        LoginAction.java
        LoginModel.java
    classes
        *.class.
        ApplicationResources.properties
    lib
        *.jar (6 jar files)
```

http://localhost:8081/programlidationloginapplication.

ApplicationResources.properties

user.input = Username can't be empty.
pwd.input = password can't be empty.

LoginForm.java

```
package com.nareshit.form;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
public class LoginForm extends ActionForm
{
  private String username;
  private String password;
  // setters & getters
public ActionErrors validate(ActionMapping mapping,
    HttpServletRequest request)
{
  ActionErrors ae = new ActionErrors();
  if (username.length() == 0)
      ae.add("username", new ActionMessage("user.input"));
  if (password.length() == 0)
  ae.add("password", new ActionMessage("Pwd.input"));
  return ae;
}
}
```

26/11/13

struts-config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="loginform"
    type="com.nareshit.form.LoginForm" />
  </form-beans>
  <action-mappings>
    <action name="loginform" path="/login"
    type="com.nareshit.action.LoginAction"
    validate="true" input="/login.jsp">
      <forward name="ok" path="/welcome.jsp" />
      <forward name="notok" path="/loginfailed.jsp" />
    </action>
  </action-mappings>
  <message-resources
    parameter="ApplicationResources" />
</struts-config>
```

login.jsp

```
<%@ taglib prefix="html"
  uri="http://struts.apache.org/tags-html" %>
<HTML>
  <BODY BGCOLOR="cyan">
    <CENTER>
      <html:form action="/login" method="POST">
    <TABLE>
    <TR>
      <TD colspan="3"><H1>LoginScreen</H1></TD>
    </TR>
```

```
<TR>
 <TD> username </TD>
 <TD> <html:text property = "username" /> </TD>
 <TD> <html:errors property = "username" /> </TD>
</TR>
<TR>
 <TD> password </TD>
 <TD> <html:password property = "password" /> </TD>
 <TD> <html:errors property = "password" /> </TD>
</TR>
<TR>
 <TD colspan = "2" >
  <html:submit> Login </html:submit> </TD>
</TR>
</html:form>
</CENTER>
</BODY>
</HTML>
```

Q) How does ActionServlet know whether validation failed or not?

⇒ org.apache.struts.action.ActionErrors object which is returned from validate() method indicates to ActionServlet whether validation is success or failure.

⇒ If ActionErrors object is empty or null, ActionServlet knows that validation is successful.

⇒ If actionErrors object has atleast one ActionMessage object, ActionServlet knows that validation is a failure & instead of invoking the action it switches the control to the input page specified using "input" tag of <action> tag.

Q) What are the limitations of programmatical validations in a Struts application?

⇒ 1) validation logic at times is very complex and time consuming. Zero support from framework.

2) validation logic written for one field is not reusable even if the same field is there in another form.

3) validation can't be applied in case of dynamic form beans.

4) Struts framework provided client side validation code (Javascript) also is not available. i.e developer is responsible for both sides validation code development.

note:- declarative validation addresses all the limitations of programmatical validation and therefore, in almost all cases declarative validation is only used in real Struts applications.

Q) How to implement declarative validation in a Struts Application (for static form bean)?

⇒ Using validator framework which is a subframework of Struts is nothing but using declarative validation in a Struts Application.

⇒ In case of declarative validation, validation requirements are explained to the framework using xml tags instead of developer developing the validation logic.

Step 1 :- develop the resource bundle with appropriate error keys & corresponding messages.

Step 2 :- develop the form-bean that extends org.apache.struts.validatorform.

⇒ don't implement validate() method.

Step 3 :- enable validation & specify the input page.

Step 4 :- place validator-rules.xml in WEB-INF (built-in file)

*Step 5 :- develop validation.xml and place it in WEB-INF

Step 6 :- enable validator framework.

Step 7 :- place an additional jarfile in lib folder.
commons-validator-1.3.1.jar.

Step 8 :- make use of <html:Errors/> tag in the input page to display validation failure messages.

27/11/13

Q. Modify the previous struts application so as to implement declarative validation?

⇒

```
declvalidationloginapplication
    login.jsp
    welcome.jsp
    loginfailed.jsp
    WEB-INF
        web.xml
        struts-config.xml
        validation.xml
        validator-rules.xml
    src
        LoginForm.java
        LoginAction.java
        LoginModel.java
    classes
        *.class
        ApplicationResources.properties
    lib
        *.jar (6+1 jarfiles)
```

http://localhost:8081/declvalidationloginapplications

ApplicationResources.properties

errors.required = {0} can't be empty.

errors.maxlength = {0} can't be less than {1} characters.

User = USERNAME

Pwd = PASSWORD

& = EIGHT

LoginForm.java

```
package com.nareshit.form;
import org.apache.struts.validator.ValidatorForm;
public class LoginForm extends ValidatorForm
{
   private String name;
   private String password;
   // setters & getters
}
```

struts-config.xml

```
<struts-config>
  <form-beans>
  ....
  </form-beans>
  <action-mappings>
    <action validate = "true"  input="/login.jsp">
    ......
    </action>
  </action-mappings>
  <message-resources parameter="ApplicationResources" />
  <plug-in className = "org.apache.struts.validator.
                        ValidatorPlugIn" />
    <set-
    <property property = "pathnames"
```

value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>

</plug-in>

</struts-config>

validation.xml

```
<form-validation>
  <formset>
    <form name="loginform">
      <field property="username" depends="required">
        <arg position="0" key="user"/>
      </field>
      <field property="password" depends="required,minlength">
        <arg position="0" key="pwd"/>
        <arg position="1" key="8" resource="false"/>
        <var>
          <var-name>minlength</var-name>
          <var-value>8</var-value>
        </var>
      </field>
    </form>
  </formset>
</form-validation>
```

Q) What is the significance of validator-rules.xml?

⇒ To make all built-in validators provided by Validator framework (sub framework of Struts) available to the struts application, validator-rules.xml is used

Q) what is the significance of validation.xml?

⇒ Validation requirements of the application are specified to the framework using validation.xml.

28/11/13

Q) Explain about the tags of validation.xml?

⇒ name attribute's value of <form> tag should match with logical name of the formbean specified in Struts configuration file.

⇒ For which form bean fields built-in validators required is specified using <form> tag.

⇒ We can have as many <form> tags as there are form beans in the application.

⇒ "depends" attribute of <field> tag takes the logical name of the built in validator. We can specify multiple values also with comma separator.

⇒ For which bean field validators are to be applied is specific using property attribute.

⇒ <arg> tags are for parametric replacement.

⇒ <var> tag is used to supply the arguement to a build in validator.

note:- Resource bundle messages can take parameters. for eg. {0}, {1}, {2} etc.
Supplying values to these parameters i.e place holders is nothing but parametric replacement.

Q) How to apply declarative validation for a dynamic form bean?

⇒ Step1:- Configure the following in struts-config.xml.

```
<form-bean name="loginform" type="org.apache.struts.validator.
    DynaValidatorForm">
    <form-property name="username" type="java.lang.String"/>
    <form-property name="password" type="java.lang.String"/>
</form-bean>
```

Step2:- In action class, typecast ActionForm reference to DynaValidatorForm and then capture the user input.

for eg:

```
DynaValidatorForm daf = (DynaValidatorForm) form;
String user = (String) daf.get("username");
String pwd = (String) daf.get("password");
```

note:- Remaining steps are from that of static form beans declarative validation.

Q) Develop a struts application in which declarative validations applied to a dynamic form-bean?

declvalidationdynaformbeanapplication.

- login.jsp
- welcome.jsp
- loginfailed.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - validation.xml
  - validator-rules.xml
- src
  - LoginAction.java
  - LoginModel.java
- classes

*.class
ApplicationResources.properties.
lib
*.jar (7 jar files).

http://localhost:8081/decvalidationdynaformbeanapplication

Q) How to make use of validator framework provided client side validation support?

Step1:- Make use of <html:javascript> tag in the input page of the struts application.

Step2:- invoke a special javascript function through event handling in the input page.

Q) How does Struts validator subframework provided clientside validation code function?

⇒ when the input jsp is executed, "javascript" tag of html library is executed. for this tag logical name of the form bean should be supplied as value (to "formName" attribute).

⇒ validator framework provided Javascript functions code is appended in line by the tag handler of "javascript" tag.

⇒ a specialised Javascript function also is generated & added. That function name is validateXXX(). "XXX" stands for the logical name of the form bean. this method calls the validation performing Javascript functions.

⇒ When user clicks on the submit button, "onsubmit" event is raised & the special function is called. Internally that function calls the Javascript functions and clientside validation is performed.

note:- If we develop login.jsp as follows for the previous 2 applications, client side validation (Javascript code provided by validator framework) code also will be available to those applications.

login.jsp

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

<HTML>
  <HEAD>
    <html:javascript formName="loginform" />
  </HEAD>
  <BODY BGCOLOR="yellow">
  .....
    <html:form action="/login" method="POST"
      onsubmit="return validateloginform(this);" >
  .....
  </HTML>
```

29/11/13

Q) Explain about action based declarative validation?

⇒ When multiple related input pages are using the same form bean and if form bean based declarative validation is implemented in the application, for some input screens with less number of fields, wrong validation error messages are displayed to the end-user.

⇒ To overcome that we go for action based validation.

login & registration use-cases in a single application.

struts-config.xml

```
<struts-config>:
  <form-beans>
    <form-bean name="userform"
      type="com.nareshit.form.UserForm" />
  </form-beans>
  <action-mappings>
    <action path="/login" name="userform"
```

```
type = "com.nareshit.controller.LoginAction"  validate = "true"
    input = "/login.jsp">
<forward name = "ok"  path = "/welcome.jsp" />
<forward name = "notok"  path = "/loginfailed.jsp" />

</action>
<action name = "userform"
 type = "com.nareshit.controller.RegisterAction"
 path = "/register"  validate = "true"
 input = "/register.jsp">
  <forward name = "success"  path = "/success.jsp" />
  <forward name = "failure"
   path = "/registrationfailed.jsp" />

</action>
</action-mappings>
<message-resources parameter = "ApplicationResources" />
<plug-in
  className = "org.apache.struts.validator.ValidatorPlugIn">
   <set-property  property = "pathnames"
    value = "/WEB-INF/validator-rules.xml,/WEB-INF/
        validation.xml" />

  </plug-ins>
</struts-config>
```

note:- 2 use cases i.e 2 input screens. using the same formbean to store the user input and form bean based declarative validation applied.

validation.xml

```
<form-validation>
  <form-set>
    <form name="userform">
    <field property="username" depends="required">
    <arg position="0" key="our.user"/>
  </field>
  <field property="password" depends="required">
  <arg position="0" key="our.pwd"/>
  </field>
  <field property="emailid" depends="required">
   <arg position="0" key="our.mail"/>
  </field>
  </form>
  </form-set>
</form-validation>
```

ApplicationResources.properties :-

```
errors.required = {0} should not be empty.
our.usr = User Name
our.pwd = Password
our.mail = Email
```

UserForm.java

```
package com.naresh.it.form; validator.ValidatorForm;
import org.apache.struts.~~action.action~~
public class UserForm extends ValidatorForm
  {
    private String username;
    private String password;
    private String emailid;
    //setters & getters
  }
```

note:- for "registerloginapplication". apply declarative validation using single form bean.

Q) How to implement action based declarative Validation?

Ans). static form bean, declarative validation, multiple input pages using the same form bean; don't want form bean based declarative validation; requires action based validation; go for org.apache.struts.validator.ValidatorActionForm.

UserForm.java

```
public class UserForm extends ValidatorActionForm
  {.....
  }
```

Validation.xml

```
<form-validation>
  <formset>
    <form name="/register">
    <field property="username" depends="required">
    <arg position="0" key="our.user" />
    </field>
      <field property="password" depends="required">
      <arg position="0" key="our.pwd" />
    </field>
      <field property="emailid" depends="required">
      <arg position="0" key="our.mailid" />.
    </field>
    </form>
    <form name="/login">
      <field property="username" depends="required">
      <arg position="0" key="our.user" />
      </field>
      <field property="password" depends="required">
    </field>
    </form>
  </formset>
</formValidation>
```

Q) How to implement action based declarative validation for static bean based application?

⇒ dynamic form bean, declarative validations, multiple input pages using the same form bean, don't want form bean based declarative validation, requires action based validation; go for org.apache.struts.validator.DynaValidatorActionForm.

struts-config.xml

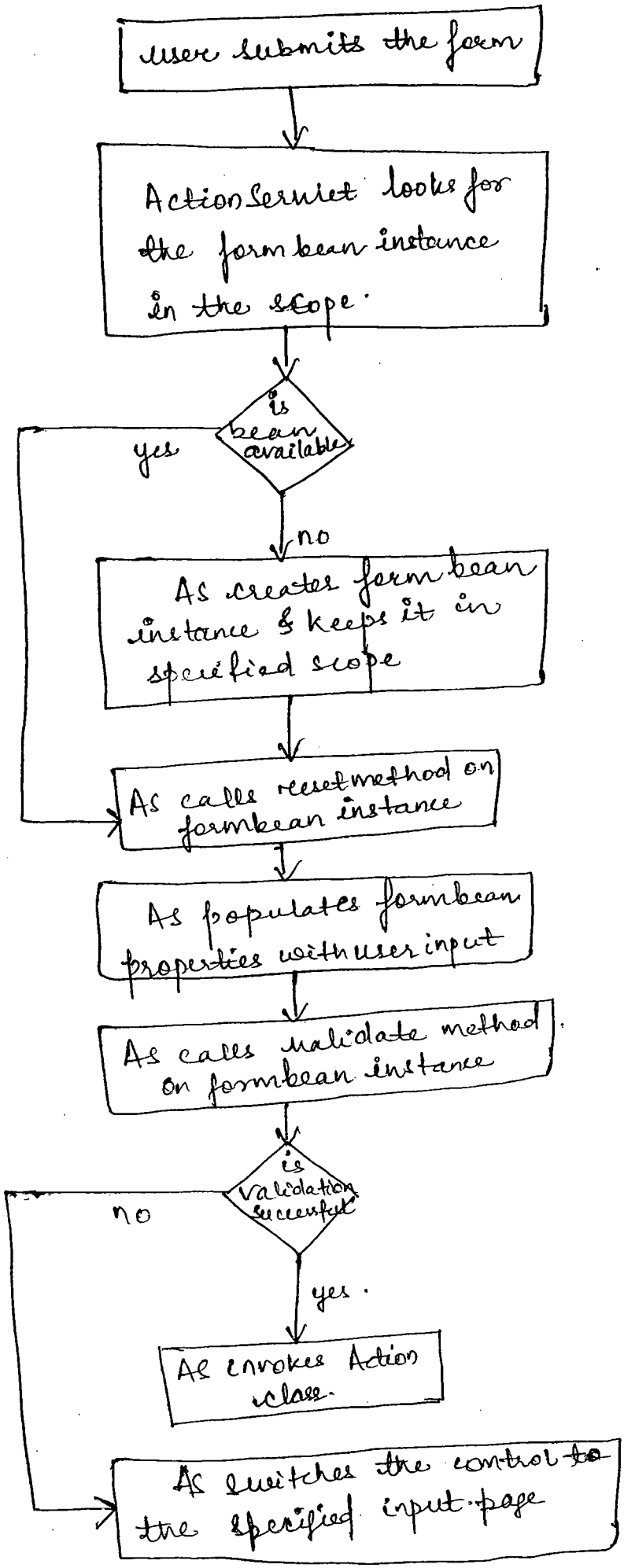
```
<struts-config>
  <form-beans>
    <form-bean name="userform"
    type="org.apache.struts.validator.DynaValidatorActionForm">
    <form-property name="username"
      type="java.lang.String" />
    <form-property name="password"
      type="java.lang.String" />
    <form-property name="emailid"
      type="java.lang.String" />
  </form-bean>
  </form-beans>
</struts-config>
```

note:- Validation.xml is from previous application.

In action classes.

```
DynaValidatorActionForm
  df = (DynaValidatorActionForm) form;
String user = (String) df.get("username");
String pwd = (String) df.get("password");
```

Q) Explain about form bean lifecycle!



08/12/13

Q) what is the purpose of reset() method in a form bean?

⇒ To overcome the strange behaviour of the browser while sending checkbox input to the server, reset() method is introduced in the form bean.

⇒ If boolean variable corresponding to the checkbox is not there in the form bean, i.e if there is no checkbox involved in the web form, no need of reset method.

⇒ Even if checkbox exists in the form, if the bean is to be kept in request scope, there is no need of reset method in the form bean class.

⇒ In reset method, boolean variable corresponding to a checkbox is set to default value i.e false.

note :- In a dynamic form bean checkbox issue is addressed using "initial" attribute of <form-property> tag.

foreg :-

```
<form-property name = "isAgreed"
    type = "java.lang.Boolean" initial="false" />
```

INTERNATIONALIZATION (i18n)

Q) what is i18n?

⇒ providing multi lingual support to the website without changing application's source code is nothing but i18n (internationalization).

⇒ Once the struts application is i18n enabled, based on user preferred language, web pages can be displayed.

Q) How to implement i18n in a Struts application?

Step 1 :- Develop as many resource bundles as the no. of languages that you want to support.

⇒ Resource bundle keys are common. only values will be in corresponding language.

Ep2:- In jsps don't write language specific text ---
instead, make use of bean library's message tag that can obtain language content from the appropriate resource bund based on user preferred language.

<bean:message key="resource bundle key" />

) Develop a struts application in which i18n is implemen ed?

```
i18napplication
    login.jsp
    welcome.jsp
    WEB-INF
      web.xml
      struts-config.xml
      src
        LoginForm.java
        LoginAction.java
        LoginModel.java
      classes
        *.class
        ApplicationResources_en.properties
        ApplicationResources_it.properties
      lib
        *.jar (6 jar files)
```

http://localhost:8081/i18napplication

login.jsp

```
<%@ taglib prefix = "html" uri = http://struts.apache.org/tags-
                                                    ml " %>

<%@ taglib prefix = "bean" uri = http://struts.apache.org/tags-bea

<html:html>
    <BODY BGCOLOR = "YELLOW">
     <CENTER><br><br/>
    <bean:message key = "welcome.message" /><BR><BR>
    <html:form action = "login" method = "post">
     <bean:message key = "username" /><html:text propen
     = "username" /><BR><BR>
    <bean:message key = "password" /><html:password
     property = "password" /><BR><BR>
    <html:submit><bean:message key = "register.submit
                                    </html:submit>

    </html:form>
  </CENTER>

</BODY>
</html:html>
```

welcome.jsp

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

<HTML>
    <BODY BGCOLOR = "cyan">
      <H2><bean:message key = "welcome" /></H2>
    </BODY>
</HTML>
```

ApplicationResources_en.properties

welcome.message = welcome to english user.

welcome = Hello english user welcome to our website.

username = username

password = password

register.submit = login

ApplicationResources_it.properties

welcome.message = Dare il benvenuto all'operatore Italiano (welcome to Italian user)

welcome = Dare il benvenuto all'operatore Italiano.

username = nome di operatore.

password = parola d'ordine.

register.submit = registero

8/12/18

Q) How does i18n work?

⇒ Browser sends user preferred language to the web server using Accept-language HTTP header.

⇒ web server passes on that information to servlet engine.

⇒ Encapsulation of (object oriented representation of) user preferred language is nothing but java.util.Locale object.

⇒ container creates locale object encapsulating user preferred language & stores it (its reference) into HttpServletRequest object.

⇒ HttpServletRequest has getLocale method that returns Locale object.

```
Locale locale = request.getLocale();
```

⇒ Struts framework gets Locale object and stores it into session scope.

⇒ message tag of bean library selects the appropriate resource bundle based on Locale object stored in the session scope.

⇒ message tag retrieves the message from the resource bundle and write enline that message into the jsp's browser stream.

Q) What is the limitation of the "i18application" (previous application?

⇒ user should be able to change the browser setting for i18 to work.

⇒ In the first page of the website user preferred language take links should be specified so that ~~in the first page of~~ without the need of changing the browser settings, user can opt for the language.

Q) How to overcome the above limitations?

⇒ using localeAction (a built-in action)

Q) How to make the use of localeAction in a struts application while implementing i18n?

⇒ org.apache.struts.action.LocaleAction is one of the built in actions provided by struts framework.

⇒ This action class facilitates i18n without the need of user changing the browser settings.

⇒ using localeAction in a struts application involves the following steps.

Step1 :- configure a special form bean to hold user specified locale details.
ie language, country, variants

Step2:- configure LocaleAction in struts-config.xml specify the logical name of above bean for LocaleAction configuration.

Step3:- create links on home page that target LocaleAction class. Specify language and page request parameters as part of query string.

Note:- for each language option, developing a resource bundle is as usual.

Q) Develop an i18n enabled Struts application that provides multilingual support for the end-users without the need of changing the browser settings?

⇒ struts-config.xml

```
<struts-config>
 <form-beans>
   <form-bean name = "loginform" type = "com.nareshit.view.LoginForm" />
   <form-bean name = "localeBean" type = "org.apache.struts.action.
                                          DynaActionForm" />

   <form-property name = "language"
     type = "java.lang.String" />
   <form-property name = "varient"
     type = "java.lang.String" />
   <form-property name = "country"
     type = "java.lang.String" />
   <form-property name = "page"
     type = "java.lang.String" />
 </form-beans>
 <action-mappings>
   <action path = "/show" forward = "/login.jsp" />
   <action type = "com.nareshit.controller.LoginAction"
```

```
path = "/login"  name = "loginform"
input = "/login.jsp" >
  <forward name = "success"
    path = "/welcome.jsp" />
<forward name = "failure" path = "/loginfailed.jsp" />
</action>
<action path = "/changelocale"
    type = "org.apache.struts.actions.LocaleAction"
    name = "localeBean" />

</action-mappings>
<message-resources parameter = "ApplicationResources" />
</struts-config>
```

login.jsp

```
<BODY BGCOLOR = "YELLOW" >
<CENTER> <br/> <br/>
<html:link
  href = "changelocale.do?language=en&country=US&page=
    /show.do" > English </html:link>

<html:link
  href = "changelocale.do?language=it&page=/show.do" >
  Italian </html:link> <BR> <BR>
<bean:message key = "welcome.message" /> <BR> <BR>
```

:

Note: directory structure from previous application & place struts-extras-1.3.10.jar file in lib folder additionally.

Q) Explain about tiles in the content of a Struts application?

⇒ Tiles is a subframework of struts.

→ Tiles overcome the limitation of JSP include mechanism.

→ Tiles facilitates composite views creation in a struts application with consistent look & feel.

→ Using tiles in a struts application envolves the following steps:-

Step1:- Develop a layout page (design page)

Step2:- Develop the content pages.

Step3:- Develop tiles-defs.xml.

Step4:- Enables tiles sub-framework.

Step5:- place an additional jar file in "lib" folder i.e struts-tiles-1.3.10.jar.

Step6:- Specify chain-config.xml to action servlet as init parameter value. Init parameter name should be "chainConfig".

Step7:- use the tiles definition.

Q) How to enable Tiles sub framework in a struts-Application?

⇒ By using <plug-in> tag in struts-config.xml

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

Q) What is the significance of tiles-defs.xml?

⇒ for each composite view of application, one definition is mentioned in this xml file. A logical name is given to each definition.

⇒ In each definition, layout page is specified & the actual jsps that are to be assembled into one composite views are also specified.

⇒ When a tiles definition is used, Tiles subframework performs the assembling of all the simples view (each tile) into a composite views around the layout page.

Q) What is the significance of chain-config.xml?

⇒ Prior to Struts 1.3, tiles sub framework was using Request processor while assembling the views. In Struts 1.3.x this xml file is assisting tiles in creating the composite views.

Q) What is the limitation of JSP include mechanism?

⇒ JSP include mechanism provides only the reusability of output generation code but not the design reusability.

⇒ If include page name changes, include mechanism totally fails.

05/12/13

Q) How to make use of tiles definition?

⇒ 1) <tiles:insert definition = "logical name" />

2) <forward name = "logical name of the path to the view" path = "logical name of the tiles definition" />

3) <action path = "/actionpath" forward = "Logical name of the tiles definition" />

⇒ Generally if home page is the composite view, first one is used.

⇒ When user submitted the form, the response page has to be the composite view means the second one is used.

⇒ When user clicked on the hyperlink the target page has to have composite view means go for third style of using the tiles definition.

Q) Develop a struts application in which tiles is used to develop the (composite) views of the website.

tilesapplication

- footercontent.jsp
- headercontent.jsp
- homebodycontent.jsp
- homevieww.jsp
- layout.jsp
- loginbodycontent.jsp
- menucontent.jsp
- menucontenttwo.jsp
- registerbodycontent.jsp
- WEB-INF
  - web.xml
  - struts-config.xml
  - tiles-defs.xml
  - lib
    - *.jar (6 jar files)
    - struts-tiles-1.3.10.jar

http://localhost:8081/tilesapplication

layout.jsp

```
<%@ taglib prefix="tiles"
uri="http://struts.apache.org/tags-tiles"%>
<html>
<body BGCOLOR="wheat">
<table border="1" width="100%" height="100%">

<tr>
<td colspan="2">
<tiles:insert attribute="header" />
</td>
</tr>

<tr>
<td width="30%"><tiles:insert attribute="menu" /></td>
<td><tiles:insert attribute="body" /></td>
</tr>
```

```
</tr>
<td colspan="2"><tiles:insert attribute="footer" /></td>
</tr>
</table></body></html>
```

tiles-defs.xml

```
<tiles-definitions>
  <definition path="/layout.jsp" name="logindef">
    <put name="header"
     value="/headercontent.jsp" />
    <put name="footer" value="/footercontent.jsp" />
    <put name="menu" value="/menucontentone.jsp" />
    <put name="body" value="/logbodycontent.jsp" />
  </definition>
  <definition name="registerdef" extends="logindef">
    <put name="body" value="/registerbodycontent.jsp" />
  </definition>
  <definition name="welcomedef" extends="logindef">
    <put name="menu" value="/menucontenttwo.jsp" />
    <put name="body" value="/homebodycontent.jsp" />
  </definition>
</tiles-definitions>
```

struts-config.xml

```
<struts-config>
  <action-mappings>
    <action path="/login" forward="logindef"/>
    <action path="/register" forward="registerdef"/>
    <action path="/userhome" forward="userhomedef"/>
  </action-mappings>
  <plug-in ClassName="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config"
      value="/WEB-INF/tiles-defs.xml"/>

  </plug-in>
</struts-config>
```

web.xml

```
<web-app>
  <servlet>
    <servlet-name>frontcontroller</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-
                                                          class>

    <init-param>
      <param-name>ChainConfig</param-name>
      <param-value>org/apache/struts/tiles/chain-config.xml
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>

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

```
<welcome-file-list>
    <welcome-file> homeview.jsp </welcome-file>
</welcome-file-list>
</web-app>
```

homeview.jsp

```
<%@ taglib prefix="tiles"
    uri="http://struts.apache.org/tags-tiles" %>
<tiles:insert definetion="logindef" />
```

06/12/13

footercontent.jsp

```
<center><i> Naresh i technologies, Ameerpet
```

headercontent.jsp

```
<MARQUEE><FONT COLOR="green" SIZE="4">WELCOME TO
    WWW.NARESHIT.COM</FONT></MARQUEE>
```

homebodycontent.jsp

```
<CENTER><FONT COLOR="red" SIZE="5">HELLO GUEST USER</FONT>
    </CENTER>
```

loginbodycontent.jsp

```
<form action="userhome.do">
User Name: <input type="text" name="username" /><br/>
password: <input type="password" name="password" />
<br/><input type="submit" name="login" />
</form>
```

menucontentone.jsp

```
<li><a href="login.do">login</a></li><br/>
<li><a href="register.do">Register</a></li><br/>
```

menucontent2.jsp

```
<li><a href="login.do">Logout</a></li>
```

registerbodycontent.jsp

```
<H1> Registeration goes here </H1>
```

Q) what is file uploading?

⇒ sending a file from client machine to webserver machine is nothing but file uploading.

Q) How to perform file uploading in a struts application?

⇒ Step1:- Develop the form bean with org.apache.struts.upload.FormFile field.

⇒ object oriented representation of user uploaded file is nothing but "FormFile" object.

Step2:- In execute method of action class, get the file details (name, content, size etc) from FormFile object & implement I/O logic to store the file into the webserver machine.

Step3:- Create user input page with the following.

1) HTTP request method is POST.

2) user control type is "file".

3) make use of the following attribute with the specified value of form tag.

enctype = "multipart/form-data"

Step4:- Additionally place commons-fileupload.jar file into lib folder.

Q) Develop a strats application to implement file uploading.

⇒ fileuploadapplication.
fileupload.jsp (input page)
success.jsp.
failure.jsp.
WEB-INF
web.xml.
struts-config.xml
src
FileUploadAction.java
FileUploadForm.java
classes
*.class
lib
*.jar (6 jarfiles)
commons-fileupload.jar

http://localhost:8081/fileuploadapplication.

fileupload.jsp.

```
<html>
<body BGCOLOR = "pink">
<center>
<h1> File Uploading screen </h1>
<form action = "fileUploadAction.do" method = "post"
enctype = "multipart/form-data">
File location <input type = "file" name = "file"><br>
<input type = "submit" value = "upload">
</form>
</center>
</body>
</html>
```

FileUploadAction.jsp

```
package com.naresh.view;
import org.apache.struts.action.ActionForm;
import org.apache.struts.upload.FormFile;
public class FileUploadForm extends ActionForm
{
   private FormFile file;
   public FormFile getfile();
    {
     return file;
   }
   public void setfile(FormFile file)
     {
       this.file = file;
     }
}
```

success.jsp

```
<html>
  <body bgcolor="pink">
   <center>
   <h1>File uploaded successfully</h1>
   </center>
  </body>
</html>
```

failure.jsp

```
<html>
 <body bgcolor="pink">
  <center>
   <h1> could not upload the file </h1>
  </center>
 </body>
</html>
```

FileUploadAction.java

```
package com.nareshit.controllers;
import java.io.FileOutputStream;
import javax.servlet.http.*;
 // org.apache.struts.action.*;
 // org.apache.struts.upload.FormFile;
 // com.nareshit.view.FileUploadForm;
public class FileUploadAction extends Action
{
public ActionForward execute(ActionMapping mapping,
ActionForm form, HttpServletRequest request, HttpServlet
 Response response)
{
  String repage = "success";
  FileUploadForm fileUploadForm = (FileUploadForm) form;
try
 {
  FormFile formFile = fileUploadForm.getFile();
  String path = getServlet().getServletContext().getRealPath("")
      + "/" + formFile.getFileName();
```

```
FileOutputStream fos = new FileOutputStream(path);
fos.write(formfile.getFileData());

fos.close();

}
catch(Exception e)
{
  rpage = "failure"
  e.printStackTrace();
}
return mapping.findForward(rpage);
}
}
```

```
accountapplication
 account.jsp
 accountdetails,jsp
 noaccount.jsp

---WEB-INF
    |   web.xml
    |
    +---classes
    |   |   struts.xml
    |   |
    |   +---com
    |       +---nareshit
    |           +---controller
    |           |       AccountAction.class
    |           |
    |           +---service
    |           |       AccountModel.class
    |           |       DBUtil.class
    |           |
    |           +---vo
    |                   Account.class
    |
    +---lib
    |       commons-logging-1.1.jar
    |       freemarker-2.3.8.jar
    |       ognl-2.6.11.jar
    |       ojdbc14.jar
    |       struts2-core-2.0.6.jar
    |       xwork-2.0.1.jar
    |
    +---src
            Account.java
            AccountAction.java
            AccountModel.java
            DBUtil.java
http://localhost:8081/accountapplication
//account.jsp
<%@ taglib  uri="/struts-tags" prefix="s" %>
<HTML>
  <BODY BGCOLOR="cyan">
       <CENTER>
       <H1>Account details retrieval</H1>
      <s:form  action="account">
   <s:textfield  label="ACCOUNT NUMBER" name="accno" /> <BR><BR>
    <s:submit value="get account details"></s:submit>
      </s:form>
       </CENTER>
 </BODY>
</HTML>
-------------------------------------
//accountdetails.jsp
<%@ taglib  prefix="s"  uri="/struts-tags"  %>
<HTML>
  <BODY BGCOLOR="cyan">
<H1>ACCOUNT DETAITLS OF A/c No <s:property   value="accno" /> </H1>
A/c HOLDER NAME:  <s:property   value="name" /> <BR>
BALANCE Rs. <s:property   value="balance" /> <BR>
  </BODY>
</HTML>
-------------------------------------
// noaccount.jsp
<HTML>
   <BODY BGCOLOR="cyan">
    <H1>Account Doesn't exist</H1>
   </BODY>
</HTML>
-------------------------------------
<web-app>
        <filter>
                <filter-name>frontcontroller</filter-name>
                    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
        </filter>
                <filter-mapping>
                    <filter-name>frontcontroller</filter-name>
                        <url-pattern>/*</url-pattern>
                </filter-mapping>
```

```
                <welcome-file-list>
                        <welcome-file>account.jsp</welcome-file>
                </welcome-file-list>
</web-app>
-------------------------------------------------
<struts>
        <package  name="mypack" extends="struts-default">
                 <action name="account" class="com.nareshit.controller.AccountAction">
                        <result  name="success">/accountdetails.jsp</result>
                        <result  name="failure">/noaccount.jsp</result>
                 </action>
        </package>
</struts>
------------------------
package com.nareshit.vo;
public class  Account implements java.io.Serializable
{
        private int accno;
        private String name;
        private float balance;
        public  void setAccno(int ano)
        {
                accno=ano;
        }
        public int getAccno()
        {
                return accno;
    }
        public void setName(String nm)
        {
                name=nm;
    }
        public String getName()
        {
                return name;
    }
        public void setBalance(float bal)
        {
                 balance=bal;
    }
        public float getBalance()
        {
                return balance;
    }
}//Value Object class  OR  Data Transfer Object class
-----------------------------------------------------
package  com.nareshit.controller;
import com.nareshit.vo.Account;
import com.nareshit.service.AccountModel;
public class  AccountAction
{
        private int accno;
        private String name;
        private float balance;
        public  void setAccno(int ano)
        {
                accno=ano;
        }
        public int getAccno()
        {
                return accno;
    }
        public void setName(String nm)
        {
                 name=nm;
    }
        public String getName()
        {
                return name;
    }
        public void setBalance(float bal)
        {
                 balance=bal;
    }
        public float getBalance()
        {
                return balance;
```

```
    }
    private static  AccountModel am;
    static
    {
            am=new  AccountModel();
    }
    public String execute()
    {
            String rpage="failure";
        Account  acc=am.getAccountDetails(accno);
        if(acc !=null)
            {
          setName(acc.getName());
                    setBalance(acc.getBalance());
                    rpage="success";
            }
            return rpage;
    }
}
-------------------------------------------------
package com.nareshit.service;
import com.nareshit.vo.Account;
import java.sql.*;
public class  AccountModel
{
    public  Account  getAccountDetails(int accno)
    {
       Account acc=null;
      Connection con=null;
       PreparedStatement ps=null;
       ResultSet rs=null;
       try
       {
            con=DBUtil.getConnection();
            ps=con.prepareStatement("SELECT * FROM ACCOUNT WHERE ACCNO=?");
            ps.setInt(1,accno);
            rs=ps.executeQuery();
            if(rs.next())
            {
                    acc=new Account();
                    acc.setAccno(rs.getInt(1));
                    acc.setName(rs.getString(2));
                    acc.setBalance(rs.getFloat(3));
            }
       }//try
       catch (SQLException e)
       {
            e.printStackTrace();
       }
       finally
            {
               DBUtil.close(con,ps,rs);
            }
       return acc;
    }
}
-----------------------------------------
package com.nareshit.service;
import java.sql.*;
public class  DBUtil
{
     static
    {
            try
            {
                    Class.forName("oracle.jdbc.driver.OracleDriver");
            }
            catch(Exception e)
            {
                    System.out.println(e);
            }
    }//static initializer
    public static  Connection getConnection() throws SQLException
    {
            return
            DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:server","scott","tiger");
    }
```

```
        public static void close(Connection con,Statement st,ResultSet rs)
        {
                try{
                        if(rs !=null)
                                rs.close();
                if(st !=null)
                                        st.close();
                        if(con !=null)
                                con.close();
                        }
                        catch(SQLException e){}
        }
}
*****************************************************************************
accountsapplication
|   account.jsp
|   noaccount.jsp
|   results.jsp
|
+---WEB-INF
    |   web.xml
    |
    +---classes
    |   |   struts.xml
    |   |
    |   +---com
    |       +---nareshit
    |           +---struts2
    |                   Account.class
    |                   AccountAction.class
    |                   AccountModel.class
    |                   DBUtil.class
    |
    +---lib
    |       commons-dbutils.jar
    |       commons-logging-1.0.4.jar
    |       freemarker-2.3.8.jar
    |       ognl-2.6.11.jar
    |       ojdbc14.jar
    |       struts2-core-2.0.6.jar
    |       xwork-2.0.1.jar
    |
    +---src
            Account.java
            AccountAction.java
            AccountModel.java
            DBUtil.java
http://localhost:8081/accountapplication
//account.jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
        <body bgcolor=cyan>
        <center>
        <H1>ACCOUNT  DETAILS RETRIEVAL SCREEN</H1>

                <s:form action="accountAction" >
                        <s:textfield label="BALANCE RANGE" name="balance"/>
                        <s:submit/>
                </s:form>
    </center>
         </body>
 </html>
---------------------------------
//noaccount.jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<HTML>
<BODY BGCOLOR="cyan">
<H1>NO ACCOUNT  EXISTS WITH BALANCE  MORE THAN Rs. <s:property value="balance" /> </H1>
</BODY>
</HTML>
-------------------------------
//results.jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<HTML>
        <BODY BGCOLOR="yellow">
                <CENTER>
                        <H1>ACCOUNT DETAILS</H1>
```

```
                        <TABLE BORDER=1 CELLPADDING=3 CELLSPACING=0>
                                <TR>
                                        <TH align=right width=100>ACCNO</TH>
                                        <TH align=right width=100>NAME</TH>
                                        <TH align=right width=100>BALANCE</TH>
                                </TR>
                                <s:iterator id="acc"  value="accounts">
                                        <TR>
                                                <TD align=right width=100><s:property value="accno"
                                                /></TD>
                                                <TD align=right width=100><s:property value="name"
                                                /></TD>
                                                <TD align=right width=100><s:property value="balance"
                                                /></TD>
                                        </TR>
                                </s:iterator>
                        </TABLE>
                </CENTER>
        </BODY>
        </HTML>
--------------------------------
<web-app>
    <filter>
        <filter-name>frontcontroller</filter-name>
        <filter-class>
          org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>
    <filter-mapping>
        <filter-name>frontcontroller</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <welcome-file-list>
        <welcome-file>account.jsp</welcome-file>
    </welcome-file-list>
</web-app>
-------------------------------------
<struts>
        <package name="mypack" extends="struts-default">
                        <action name="accountAction" class="com.nareshit.struts2.AccountAction">
                                <result name="success">/results.jsp</result>
                                <result name="error">/noaccount.jsp</result>

                        </action>
        </package>
</struts>
-------------------------------------
package com.nareshit.struts2;
import java.util.*;
public class AccountAction
{
    private float balance;
        private static AccountModel model;
        private List<Account> accounts;
        static
        {
                model=new AccountModel();
        }
        public void setBalance(float balance)
        {
                this.balance=balance;
    }
        public float getBalance()
        {
                return balance;
    }
        public List<Account> getAccounts()
        {
                return accounts;
        }
        public String execute() throws Exception
        {
                  accounts=model.getAccounts(balance);
                       if(accounts!=null)
                   return  "success";
                return   "error";
        }
```

```
}
----------------------------
package com.nareshit.struts2;
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import java.util.*;
public class AccountModel
{
        public List<Account>  getAccounts(float bal)
        {
                List<Account> accounts=null;
                Connection con=null;
                PreparedStatement ps=null;
                ResultSet rs=null;
                try
                {
                        con=DBUtil.getConnection();
                        ps=con.prepareStatement("SELECT * FROM ACCOUNT WHERE BALANCE>?");
            ps.setFloat(1,bal);
                        rs=ps.executeQuery();
                        if(rs.next())
                        {
                                accounts=new ArrayList();
                                do
                                {
                                Account acc=new Account();
                                acc.setAccno(rs.getInt(1));
                                acc.setName(rs.getString(2));
                                acc.setBalance(rs.getFloat(3));
                                accounts.add(acc);
                          }while(rs.next());
                        }//if
                }//try
                catch (Exception e)
                {
                     e.printStackTrace();
                }
            finally
                {
                        DBUtil.close(con,ps,rs);
                }
                return accounts;
        }//getAccount method
}
-----------------------------------------------------
declarativeexpapplication
|    account.jsp
|    accountdetails.jsp
|    noaccount.jsp
|    problem.jsp
|
+---WEB-INF
|    |   web.xml
|    |
|    +---classes
|    |   |   struts.xml
|    |   |
|    |   +---com
|    |       +---nareshit
|    |           +---controller
|    |           |       AccountAction.class
|    |           |
|    |           +---exception
|    |           |       AccountNotFoundException.class
|    |           |       ProcessingException.class
|    |           |
|    |           +---service
|    |           |       AccountModel.class
|    |           |       DBUtil.class
|    |           |
|    |           +---vo
|    |                   Account.class
|    |
|    +---lib
|    |       commons-logging-1.1.jar
|    |       freemarker-2.3.8.jar
```

```
        |       ognl-2.6.11.jar
        |       ojdbc14.jar
        |       struts2-core-2.0.6.jar
        |       xwork-2.0.1.jar
http://localhost:8081/declarativeexpapplication
//account.jsp
<%@ taglib  uri="/struts-tags" prefix="s" %>
<HTML>
  <BODY BGCOLOR="cyan">
       <CENTER>
       <H1>Account details retrieval</H1>
      <s:form  action="account">
    <s:textfield  label="ACCOUNT NUMBER" name="accno" /> <BR><BR>
     <s:submit value="get account details"></s:submit>
      </s:form>
       </CENTER>
 </BODY>
</HTML>
------------------------------
//accountdetails.jsp
<%@ taglib  prefix="s"  uri="/struts-tags"  %>
<HTML>
  <BODY BGCOLOR="cyan">
<H1>ACCOUNT DETAITLS OF A/c No <s:property   value="accno" /> </H1>
A/c HOLDER NAME:  <s:property   value="name" /> <BR>
BALANCE Rs. <s:property   value="balance" /> <BR>
  </BODY>
</HTML>
-------------------------
<HTML>
   <BODY BGCOLOR="cyan">
    <H1>Account Doesn't exist</H1>
   </BODY>
</HTML>
-------------------------
//problem.jsp
<HTML>
   <BODY BGCOLOR="cyan">
   <FONT  COLOR="red"  SIZE="5">
    Server problem. Please try later
       </FONT>
   </BODY>
</HTML>
-----------------------------------
<web-app>
        <filter>
                    <filter-name>frontcontroller</filter-name>
                        <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
        </filter>
               <filter-mapping>
                   <filter-name>frontcontroller</filter-name>
                       <url-pattern>/*</url-pattern>
                </filter-mapping>
                <welcome-file-list>
                                <welcome-file>account.jsp</welcome-file>
                </welcome-file-list>
</web-app>.
---------------------------------------
<struts>
       <package  name="mypack" extends="struts-default">
                <action name="account" class="com.nareshit.controller.AccountAction">
                                <exception-mapping
                                exception="com.nareshit.exception.AccountNotFoundException"
                                result="noacc" />
                                    <exception-mapping
                                    exception="com.nareshit.exception.ProcessingException"
                                    result="problem" />
                                <result  name="success">/accountdetails.jsp</result>
                                    <result  name="noacc">/noaccount.jsp</result>
                                    <result  name="problem">/problem.jsp</result>
                           </action>
       </package>
</struts>
-----------------------------------------------
package  com.nareshit.controller;
import com.nareshit.vo.Account;
import com.nareshit.service.AccountModel;
```

```
public class  AccountAction
{
        private int accno;
        private String name;
        private float balance;
        private static  AccountModel am;
        public  void setAccno(int ano)
        {
                accno=ano;
        }
        public int getAccno()
        {
                return accno;
    }
        public void setName(String nm)
        {
                 name=nm;
    }
        public String getName()
        {
                return name;
    }
        public void setBalance(float bal)
        {
                 balance=bal;
    }
        public float getBalance()
        {
                return balance;
    }
        static
        {
                am=new  AccountModel();
        }
        public String execute() throws Exception
        {
                Account  acc=am.getAccountDetails(accno);
        setName(acc.getName());
                setBalance(acc.getBalance());
                return "success";
        }
}
---------------------------------
package com.nareshit.service;
import com.nareshit.vo.Account;
import java.sql.*;
import com.nareshit.exception.AccountNotFoundException;
import com.nareshit.exception.ProcessingException;

public class  AccountModel
{
        public  Account  getAccountDetails(int accno) throws
        AccountNotFoundException,ProcessingException
        {
           Account acc=null;
         Connection con=null;
           PreparedStatement ps=null;
           ResultSet rs=null;
           try
           {
                   con=DBUtil.getConnection();
                   ps=con.prepareStatement("SELECT * FROM ACCOUNT  WHERE ACCNO=?");
                   ps.setInt(1,accno);
                   rs=ps.executeQuery();
                   if(rs.next())
                   {
                           acc=new Account();
                           acc.setAccno(rs.getInt(1));
                           acc.setName(rs.getString(2));
                           acc.setBalance(rs.getFloat(3));
                   }
                   else
                           throw new AccountNotFoundException();
           }//try
           catch (SQLException e)
           {
                   throw new ProcessingException();
```

```
            }
            finally
                {
                    DBUtil.close(con,ps,rs);
                }
            return acc;
        }
}
-----------------------------------------
annotationvalidationapplication
|   failure.jsp
|   login.jsp
|   mystyle.css
|   welcome.jsp
|
+---WEB-INF
|   |   web.xml
|   |
|   +---classes
|   |   |   struts.xml
|   |   |
|   |   +---edu
|   |       +---nit
|   |           +---struts2
|   |                   LoginAction.class
|   |
|   +---lib
|   |       commons-logging-1.1.jar
|   |       freemarker-2.3.8.jar
|   |       ognl-2.6.11.jar
|   |       struts2-core-2.0.6.jar
|   |       xwork-2.0.1.jar
|   |
|   +---src
|           LoginAction.java
http://localhost:8081/annotationvalidationapplication
//mystyle.css
.errorMessage {
        color: #CB3216;
        font-size: 20px;
}
----------------------
//login.jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
                <title>Validation Using Annotation - Login</title>
                <link rel="stylesheet" href="mystyle.css" type="text/css" />
        </head>
        <body bgcolor=cyan>
        <center>
        <h1>Login Screen</h1>
              <s:actionerror />
                <s:form action="loginAction" validate="true">
                        <s:textfield label="Name" name="username"/>
                        <s:password label="password" name="password"/>
                        <s:submit/>
                </s:form>
    </center>
         </body>
 </html>
-----------------------------------------------------------
package edu.nit.struts2;
import com.opensymphony.xwork2.validator.annotations.*;
import com.opensymphony.xwork2.*;
public class LoginAction extends ActionSupport
{
        String username;
        String password;
        @RequiredFieldValidator(fieldName = "username", message = "User Name field is empty.")
        public void setUsername(String name)
        {
                username = name;
        }
        public String getUsername()
        {
                return username;
```

```
        }
        @RequiredFieldValidator(fieldName = "password", message = "Password field is empty.")
        public void setPassword(String password)
        {
                this.password=password;
        }
        public String getPassword()
        {
                return password;
        }
        public String execute() throws Exception
        {
                    if(getUsername().equals(getPassword()))
                      return  SUCCESS;
                 else
                        return ERROR ;
        }
}
----------------------
<struts>
        <package name="login" extends="struts-default">
                <action name="loginAction" class="edu.nit.struts2.LoginAction">
                                <result name="success">/welcome.jsp</result>
                                <result name="input">/login.jsp</result>
                                <result name="error">/failure.jsp</result>

                </action>
        </package>
</struts>
************************************************
declarativevalidationapplication
|   failure.jsp
|   login.jsp
|   mystyle.css
|   welcome.jsp
|
+---WEB-INF
    |   web.xml
    |
    +---classes
    |   |   struts.xml
    |   |
    |   +---edu
    |       +---nit
    |           +---struts2
    |                   LoginAction-validation.xml
    |                   LoginAction.class
    |
    +---lib
    |       commons-logging-1.1.jar
    |       freemarker-2.3.8.jar
    |       ognl-2.6.11.jar
    |       ojdbc14.jar
    |       struts2-core-2.0.6.jar
    |       xwork-2.0.1.jar
    |
    +---src
            LoginAction.java
http://localhost:8081/declarativevalidationapplication
package edu.nit.struts2;
import com.opensymphony.xwork2.*;
public class LoginAction extends ActionSupport
{
        String username;
        String password;
        public void setUsername(String name)
        {
                username = name;
        }
        public String getUsername()
        {
                return username;
        }
        public void setPassword(String password)
        {
                this.password=password;
        }
```

```
        public String getPassword()
        {
                return password;
        }
        public String execute() throws Exception
        {
                    if(getUsername().equals(getPassword()))
                      return  SUCCESS;
                 else
                          return ERROR ;
        }
}
-------------------------------------------------------------
<struts>
        <package name="login" extends="struts-default">
                <action name="loginAction" class="edu.nit.struts2.LoginAction">
                                <result name="success">/welcome.jsp</result>
                                <result name="input">login.jsp</result>
                                <result name="error">/failure.jsp</result>
                </action>
        </package>
</struts>
---------------------------------------
//LoginAction-validation.xml
<validators>
        <field name="username">
                <field-validator type="requiredstring">
                        <message>user name  is required</message>
                </field-validator>
        </field>
        <field name="password">
                <field-validator type="requiredstring">
                        <message>Password field is empty.</message>
                </field-validator>
                <field-validator type="stringlength">
                        <param name="maxLength">10</param>
                        <param name="minLength">4</param>
                        <param name="trim">true</param>
                        <message>Enter password 4-10 characters long.</message>
                </field-validator>
        </field>
</validators>
------------------------
//login.jsp
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
                <title>Validation declaratively</title>
                <link rel="stylesheet" href="mystyle.css" type="text/css" />
        </head>
        <body bgcolor=cyan>
        <center>
        <h1>Login Screen</h1>
                <s:form action="loginAction">
                        <s:textfield label="Name" name="username"/>
                        <s:password label="password" name="password"/>
                        <s:submit/>
                </s:form>
    </center>
         </body>
 </html>
*************************************************
i18napplication
|   login.jsp
|   welcome.jsp
+---WEB-INF
    |   web.xml
    +---src
    |       IndexAction.java
    |       LoginAction.java
    +---lib
    |       commons-logging-1.0.4.jar
    |       freemarker-2.3.8.jar
    |       ognl-2.6.11.jar
    |       struts2-core-2.0.6.jar
    |       xwork-2.0.1.jar
    +---classes
```

```
        |   struts.properties
        |   struts.xml
        +---edu
            +---nit
                +---struts2
                        IndexAction.class
                        LoginAction.class
                        ResourceBundle_de.properties
                        ResourceBundle_en.properties
                        ResourceBundle_fr.properties
http://localhost:8081/struts2i18n
//login.jsp
<%@ taglib uri="/struts-tags" prefix="s" %>
<html>
        <body bgcolor=cyan>
        <center>
        <h2 align="center">
 <s:text name="app.change"/> |
         <a href="index.action?request_locale=de">German</a> |
         <a href="index.action?request_locale=en">English</a> |
     <a href="index.action?request_locale=fr">French</a> |
      <br><br>
        <hr>
            <s:actionerror />
                <s:form action="loginAction" method="post">
                        <s:textfield name="username" key="app.username" size="15"  />
                   <s:password name="password" key="app.password" size="15"/>
                   <s:submit key="app.submit"/>
                </s:form>
    </center>
        </body>
 </html>
 ----------------------------------
 //welcome.jsp
 <%@ taglib prefix="s" uri="/struts-tags" %>
<html>
        <body BGCOLOR=yellow>
   <h2><s:text name="app.success_login"/></h2>
   <a href="index.action"><s:text name="app.logout"/></a>
                </body>
</html>
--------------------------------
<web-app>
<filter>
     <filter-name>struts2</filter-name>
     <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
 </filter>
 <filter-mapping>
     <filter-name>struts2</filter-name>
     <url-pattern>/*</url-pattern>
 </filter-mapping>
 <welcome-file-list>
      <welcome-file>login.jsp</welcome-file>
 </welcome-file-list>
</web-app>
------------------------------
package edu.nit.struts2;
import com.opensymphony.xwork2.ActionSupport;
public class IndexAction extends ActionSupport
{
        public String execute() throws Exception
        {
                return SUCCESS;
        }
}
----------------------------------
package edu.nit.struts2;
import com.opensymphony.xwork2.*;
public class LoginAction extends ActionSupport
{
        String username;
        String password;
        public void setUsername(String name)
        {
                username = name;
        }
        public String getUsername()
```

```
914         {
915                 return username;
916         }
917         public void setPassword(String password)
918         {
919                 this.password=password;
920         }
921         public String getPassword()
922         {
923                 return password;
924         }
925         public String execute() throws Exception
926         {
927                         if(getUsername().equals(getPassword()))
928                           return  SUCCESS;
929                         else
930                         {
931                                         this.addActionError(getText("app.invalid"));
932                                         return ERROR ;
933                          }
934         }
935         public void validate()
936         {
937                 if(getPassword().length()==0)
938                         addFieldError("password",getText("app.password.blank"));
939                 if(getUsername().length()==0)
940                         addFieldError("username",getText("app.username.blank"));
941         }
942 }
943 ------------------------------
944 //ResourceBundle_de.properties
945 app.username=Benutzer Name
946 app.password=Kennwort
947 app.success_login=Willkommen! Du hast erfolgreich angemeldet.
948 app.username.blank=Benutzer-Name wird angefordert
949 app.password.blank=Kennwort wird angefordert
950 app.invalid=Unzulässiger Benutzer-Name oder Kennwort.
951 app.submit=Login
952 app.logout=Abmelden
953 app.change=Sprache ändern
954 ------------------------------
955 //ResourceBundle_en.properties
956 app.username=User Name
957 app.password=Password
958 app.success_login=Welcome! You have logged in Successfully.
959 app.username.blank=User Name is Required
960 app.password.blank=Password is Required
961 app.submit=Login
962 logout=Logout
963 app.invalid=Invalid User Name or Password.
964 app.change=Change Language
965 ------------------------------------
966 //ResourceBundle_fr.properties
967 app.username=Nomd'utilisateur
968 app.password=Mot de passe
969 app.success_login=Bienvenue ! Vous avez entré avec succès.
970 app.username.blank=Le nom d'utilisateur est exigé
971 app.password.blank=Le mot de passe est exigé
972 app.invalid=Nom ou mot de passe inadmissible d'utilisateur
973 app.submit=Ouverture
974 logout=Déconnexion
975 ------------------------------------
976 //struts.properties
977 struts.custom.i18n.resources=edu.nit.struts2.ResourceBundle
978 ---------------------------------------------------------------------
979 //struts.xml
980 <struts>
981         <package name="i18npack" extends="struts-default">
982                 <action name="loginAction" class="edu.nit.struts2.LoginAction">
983                         <result name="success">/welcome.jsp</result>
984                         <result name="error">/login.jsp</result>
985                         <result name="input">/login.jsp</result>
986                 </action>
987                 <action name="index" class="edu.nit.struts2.IndexAction">
988                         <result name="success">login.jsp</result>
989                 </action>
990         </package>
```

```
</struts>
*********************************************
tilesapplication
|   farewelbodycontent.jsp
|   footercontent.jsp
|   headercontent.jsp
|   home.jsp
|   layout.jsp
|   loginbodycontent.jsp
|   menucontent.jsp
+---WEB-INF
    |   tiles.xml
    |   web.xml
    +---src
    |       LogoutAction.java
    +---lib
    |       commons-beanutils-1.7.0.jar
    |       commons-digester-1.8.jar
    |       commons-logging-api-1.1.jar
    |       freemarker-2.3.8.jar
    |       ognl-2.6.11.jar
    |       struts2-core-2.0.6.jar
    |       struts2-tiles-plugin-2.0.6.jar
    |       tiles-api-2.0.1.jar
    |       tiles-core-2.0.1.jar
    |       xwork-2.0.1.jar
    +---classes
        |   struts.xml
        +---com
            +---nit
                +---struts2
                        LogoutAction.class
http://localhost:8081/tilesapplication
//home.jsp
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<tiles:insertDefinition   name="homedef" />
-----------------------------------------------
//headercontent.jsp
<MARQUEE><FONT COLOR="green" SIZE="6">WELCOME TO WWW.NIT.EDU</FONT></MARQUEE>
---------------------------------
//layout.jsp
<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<html>
        <body BGCOLOR="wheat">
                <table border="1" width="100%" height="100%">
                <tr>
                        <td colspan="2">
                                <tiles:insertAttribute name="header"/>
                        </td>
                </tr>
                <tr>
                        <td width="30%"><tiles:insertAttribute name="menu"/> </td>
                    <td><tiles:insertAttribute name="body"/></td>
                </tr>
                <tr>
                        <td colspan="2"><tiles:insertAttribute name="footer"/></td>
                </tr>
        </table>
   </body>
</html>
------------------------------
//menucontent.jsp
<li><a href="logout.action">Logout</a></li>
-----------------------------------------------
//loginbodycontent.jsp
<H1>LOGIN SCREEN COMES HERE</H1>
-----------------------------------------
//farewelbodycontent.jsp
<H1>THANK YOU VISIT AGAIN</H1>
-----------------------------------------------
//footercontent.jsp
<center><i><FONT COLOR="green" SIZE="6">Naresh i technologies,Ameerpet</FONT></i></center>
-------------------------------------------
//tiles.xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE tiles-definitions PUBLIC
        "-//Apache Software Foundation//DTD Tiles Configuration 2.0//EN"
```

```
        "http://tiles.apache.org/dtds/tiles-config_2_0.dtd">
<tiles-definitions>
 <definition name="homedef" template="/layout.jsp">
       <put-attribute name="header" value="/headercontent.jsp"/>
      <put-attribute name="menu" value="/menucontent.jsp"/>
      <put-attribute name="body"   value="/loginbodycontent.jsp"/>
      <put-attribute name="footer" value="/footercontent.jsp"/>
  </definition>
  <definition name="logoutdef" extends="homedef">
        <put-attribute name="body"   value="/farewelbodycontent.jsp"/>
   </definition>
</tiles-definitions>
------------------------------
//web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <filter>
     <filter-name>struts2</filter-name>
     <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
 </filter>
 <filter-mapping>
     <filter-name>struts2</filter-name>
     <url-pattern>/*</url-pattern>
 </filter-mapping>
  <listener>
  <listener-class>org.apache.struts2.tiles.StrutsTilesListener</listener-class>
 </listener>
 <welcome-file-list>
  <welcome-file>home.jsp</welcome-file>
 </welcome-file-list>
</web-app>
------------------------------------
//struts.xml
<struts>
<package name="default" extends="tiles-default">
                <action name="logout" class="com.nit.struts2.LogoutAction">
                                <result name="farewel" type="tiles">logoutdef</result>
                </action>
        </package>
</struts>
------------------------------
//LogoutAction.java
package com.nit.struts2;
public class LogoutAction
{
            public String execute() throws Exception
            {
                        return  "farewel";
             }
}
```

```
FileOutputStream fos = new FileOutputStream(path);
fos.write(formfile.getFileData());
fos.close();
}
catch(Exception e)
{ rpage = "failure"
  e.printStackTrace();
}
return mapping.findForward(rpage);
}
}
```

09/12/13

Q: How to implement multi action controller (sub controller) in a Struts application ?

→ To unify multiple related use-cases into one action class, Struts Framework has given the following built-in action classes.

1. DispatchAction
2. LookupDispatchAction
3. MappingDispatchAction
4. EventDispatchAction

Q: How to make use of DispatchAction in a Struts Application ?

Step1: Develop a user defined action class that extends Org.apache.struts.actions.DispatchAction

Step2: In the action class, implement as many user defined methods as there are use-cases to unify (Combine).

→ User defined method name can be anything but its prototype should resemble that of execute.

→ Should not implement execute() method in the user defined action class.

Step3: Configure the user defined action class as usual in Struts configuration file with an additional attribute "Parameter". We can give any user defined String as value to this attribute.

Step4: Create input pages by observing the following things

1. all the input screens point to the same action (Path)
2. request Parameter name of the webform should be that of "parameter" attribute value.
3. request Parameter value of the webform i.e Submit button caption should be user defined method name of the corresponding use case implemented in the action class.

Q. Develop a Struts application in which, multiple usecases are implemented in a single action class using DispatchAction.

~~dispatcher~~ dispatchactionapplication

- Storeemp.jsp
- deleteemp.jsp
- Success.jsp
- failure.jsp
- WEB-INF
  - web.xml
  - Struts-Config.xml
- src
  - EmployeeAction.java
  - EmployeeForm.java
  - EmployeeModel.java
- Classes
  - *.class

http://localhost:8081/dispatchapplication

Employeeform.java

```
Package com.nareshit.form;
import org.apache.struts.action.ActionForm;
Public Class Employeeform extends ActionForm
{ Preivate int empno;
  Preivate String name;
  Preivate float ~~balance~~ Sapary;
  // Setters & getters
}
```

Struts-config.xml

```
<Struts-config>
  <form-beans>
    <form-bean name="employeeform"
      type="com.nareshit.form.EmployeeForm"/>
  </form-beans>
  <global-forwareds>
    <forward name="success" path="/Success.jsp"/>
    <forward name="failure" path="/failure.jsp"/>
  </global-forwareds>
  <action-mappings>
    <action name="employeeform" path="/emp"
    type="Com.nareshit.action.EmployeeAction" Parameter="empTask"/>
  </action-mappings>
</Struts-config>
```

EmployeeAction.java

```
Package com.nareshit.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.apache.struts.actions.DispatchAction;
```

```
import com.nareshit.model.EmployeeModel;
import com.nareshit.form.EmployeeForm;
Public Class EmployeeAction extends DispatchAction
  { EmployeeModel em = new EmployeeModel();
Public ActionForeward appointEmployee(ActionMapping mapping, Ac
form, HttpServletRequest request, HttpServletResponse response)
  {  EmployeeForm ef = (EmployeeForm)form;
     int empno = ef.getEmpno();
     String name = ef.getName();
     float salary = ef.getSalary();
   boolean flag = em.StoreEmployee(empno, name, salary);
   if (flag)
      return mapping.findForeward("success");
      return mapping.findForward("failure");
  }
Public ActionForeward fireEmployee(- ---
  {  EmployeeForm ef = (EmployeeForm)form;
     int empno = ef.getEmpno();
  boolean flag = em.deleteEmployee(empno);
  if (flag)
      return mapping.findForward("success");
      return mapping.findForward("failure");
  }
}
```

storeemp.jsp

10.12.13

```
<%@ taglib prefix="html" Uri="http://Strouts.apache.org/tags-html" %>
<html>
 <Body BGCOLOR="cyan">
  <Centere>
     <H1> Appoint Employee </H1>
<html:form action="/emp" Method="POST">
Empno<html:text property="empno" value=""/>
 <BR><BR>
 name <html:text Property="name" value=""/>
  <BR><BR>
 Salary <html:text Property="Salary" value=""/>
  <BR><BR>
 <html:Submit Property="empTask">appointEmployee</html:Submit>
                                                  </html:form>
  </Centere>
   </Body>
    </html>
```

deleteemp.jsp

```
<%@ taglib prefix="html" uri="http://Strouts.apache.org/tags-html" %>
<html>
 <Body bgcolor="cyan">
 <Centere>
   <H1> fire Employee </H1>
 <html:form action="/emp" method="POST">
 Empno <html:text Property="empno" value=""/>
    <BR><BR>
 <html:Submit property="empTask">fireEmployee </html:Submit>
 </html:form>
  </Center>
  </Body>
  </html>
```

EmployeeModel.java

```
Paekage com.nareshit.model;
Public Class EmployeeModel
{ Public boolean StoreEmployee(int empno, String name, float Salary)
  { boolean flag= false;
    if (empno>1000)
      flag = true;
    // to write JDBC Code to insert the record
      return flag;
  }
Public boolean deleteEmployee (int empno)
  { boolean flag = false;
    if (empno>1000)
      flag = true;
    // to write JDBC Code to delete the record.
      return flag;
  }
}
```

Web.xml :

```
. . . .
. . . .
<welcome-file-list>
  <welcome-file> storeemp.jsp </welcome-file>
  <welcome-file> deleteemp.jsp </welcome-file>
</welcome-file-list>
```

Success.jsp

```
<H1> Employee task Performed Successfully </H1>
```

failure.jsp

```
<H1> Employee operation failed </H1>
```

NOTE: Place 3 jar files in Classpath before compiling the actionClass

1. Servlet-api.jar 2. Struts-core-1.3.10.jar & Struts-extras-1.3.10.jar

Q: How does DispatchAction function?

→ When ActionServlet Creates the user defined action class instance, it inherits execute method of DispatchAction class.

→ ActionServlet Calls that execute method.

→ Based on the submit button caption, execute method knows to call the corresponding user defined method.

→ Remaining flow as usual.

Q: What are the limitations of DispatchAction?

1. for all the use-cases, action should use the same form bean.
2. Can't support i18n

Q: Explain about LookupDispatchAction?

→ Org.apache.struts.actions.LookupDispatchAction is a subclass of DispatchAction. It is also one of the built-in actions of Struts.

→ Its purpose is to unify multiple use-cases into a single action class with i18n support.

→ Making use of LookupDispatchAction involves the following steps

Step1: Develop resource bundles with appropriate key, value pairs.

Step2: Develop a Java class that extends LookupDispatchAction.

Step3: Implement as many user defined methods as there are use-cases to unify.

→ Method prototype should be that of execute method except name.

Note: Shouldnot override execute() method like in DispatchAction.

Step4: Implement getKeyMethodMap() in the user defined action class

→ In this method create HashMap object.

→ Number of entries in HashMap is equal to the number of user defined methods implemented in the action class.

→ Each entry in HashMap is as follows

Key :- Resource bundle key

Value :- user defined method name

Step5: Configure the action class in Struts-config.xml with "parameters" as additional attribute in <action> tag. give as user defined name as value to that attribute.

Step6: Observe the following in input page development

a. all input screens should point to the same action class (path)

b. request parameters name of each input page should be that is, the name given as value to "parameters" attribute of the <action> tag.

c. give caption to the submit buttons of each input page using <bean:message>

Q: How does LookupDispatchAction function?

→ Its execute method performs the following things

1. Captures the submit button caption.
2. Performs reverse lookup in appropriate resource bundle to get the key corresponding to submit button caption.
3. Calls getKeyMethodMap() method and searches for the user defined java method in Map object by using resource bundle key as search criteria.

4. invokes the user defined method of user defined action.

11/12/13

Q: Develop a Struts application in which multiple use cases are unified in an action class using :LookupDispatchAction.

lookupdispatchactionapplication
- Storeemp.jsp
- deleteemp.jsp
- Success.jsp
- failure.jsp
- WEB-INF
  - web.xml
  - Struts-Config.xml
  - Src
    - EmployeeAction.java
    - EmployeeForm.java
    - EmployeeModel.java
    - Classes
      - *.Class
    - ApplicationResources_it.Properties
    - ApplicationResources_en.Properties
    - lib
      - *.jar (6+1)

http://localhost:8081/lookupdispatchactionapplication.

ApplicationResources_en.Properties

insert = Store employee (English)
dlt = delete employee (English)
Sresponse = employee task Successful (English)
fresponse = employee task failed (English)

ApplicationResources_it.Properties

insrt = Store employee (Italian)
dlt = delete employee (Italian)
Sresponse = employee task Successful (Italian)
fresponse = employee task failed (Italian)

EmployeeAction.java

```
.....
import org.apache.struts.actions.LookupDispatchAction;
Public class EmployeeAction extends LookupDispatchAction
{
  Public ActionForward appointEmployee(---)
    {
    }.....
    }
  Public ActionForward fireEmployee(---)
    {...
    }
  Public Map getKeyMethodMap()
    {
      Map map = new HashMap();
      map.put("insrt", "appointEmployee");
      map.put("dlt", "fireEmployee");
      return map;
    }
}// Action Class
```

Storeemp.jsp

```
<%@ taglib prefix="bean" uri="http://Struts.apache.org/tags-bean"
.....
<html:submit property="empTask"><bean:message key="insrt"/>
.....                                              </html:su
```

deleteemp.jsp

```
<%@
<html:submit property="empTask"><bean:message key="dlt"/>
.....
```

```
<%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean" %>
<html>
  <Body Bgcolor="wheat">
    <H1>
      <bean:message key="sresponse" />
    </H1>
  </Body>
</HTML>
```

failure.jsp

```
<%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean" %>
  <html>
    <Body BGCOLOR="Wheat">
      <H1>
        <bean: message key="fresponse" />
      </H1>
    </body>
  </html>
```

Struts-config.xml

→ Make entry for resource bundle

Q: What is the limitation of LookupDispatchAction?

→ We can't use multiple form beans for the use-cases. ie all the use-cases should use the same form bean.

→ Make use of MappingDispatchAction to overcome this limitation (This is introduced in Struts 1.2 and LookupDispatchAction was introduced prior to 1.2 version)

Q: How to make use of MappingDispatchAction in a Struts Application?

Step1: Develop a user defined action class that extends org.apache.struts.actions.MappingDispatchAction.

Step2: Implement as many user defined methods as there are tasks to unify in the action class. Don't override execute() method.

Step3: Configure the action class in struts-config.xml with Parameter as additional attribute. Attribute value to be the user defined method name. for each use-case, configure action class once & we can configure actionclass multiple times. Therefore, we get the chance to use different form for different usecases.

Step4: Create input pages with the following observations.

a. Submit button request Parameter name can be anything. But value (caption) should be user defined method.

b. action path has to be the one which is configured individually in the configuration file.

Note: Client request is directly mapped to user defined meth and hence the name for the action.

→ MappingDispatchAction doesnot support i18n.

12/12/13

Q: Develop a Struts Application in which, MappingDispatchAction is used to unify multiple use-cases into one action class

MappingDispatchActionapplication

- Storeemp.jsp
- deleteemp.jsp
- Success.jsp
- failure.jsp

WEB-INF
- Web.xml
- Struts-config.xml
- Src
  - Employeeaction.java
  - Employeeform.java
  - EmployeeModel.java
  - Classes
    - * class
  - lib
    - *.jar (6+1)

http://localhost:8081/mappingdispatchactionapplication

Struts-config.xml

```
<Struts-config>
  <form-beans>
    <form-bean name="employeeform" type="com.nareshit.form.EmployeeForm" />
    <form-bean name="employeeform2" type="org.apache.Struts.action.DynaActionForm">
      <form-property name="empno" type="java.lang.Integer" />
    </form-bean>
  </form-beans>
  <global-forwards>
    <forward name="Success" path="/success.jsp" />
    <forward name="failure" path="/failure.jsp" />
  </global-forwards>
  <action-mappings>
    <action name="employeeform" path="/storeemp"
      type="com.nareshit.action.EmployeeAction"
      parameter="appointEmployee" />
    <action name="employeeform2" path="/deleteemp"
      type="com.nareshit.action.EmployeeAction"
      parameter="fireEmployee" />
  </action-mappings>
</Struts-config>
```

Storeemp.jsp (from dispatchactionapplication)

```
<html:form action="/Storeemp" method="post">
.....
<html:submit>appointEmployee</html:submit>
```

deleteemp.jsp (from dispatchactionapplication)

```
<html:form action="/deleteemp" method="post">
.....
<html:submit>fireEmployee</html:submit>
```

EmployeeAction.java

```
..... ..... import org.apache.struts.actions.MappingDispatchAction;
Public class EmployeeAction extends MappingDispatchAction
{
....
  Public ActionForward fireEmployee(...)
  {
    DynaActionForm daf = (DynaActionForm)form;
      int empno = daf.get("empno");
    ...
  }

  Public ActionForward appointEmployee(...)
  {
  }
  ...
}
```

// multiaction Controller

Note: all other files from "dispatchactionapplication".

Q* Explain about EventDispatchAction.

→ It is one of the built-in actions of Struts.

→ Org.apache.struts.actions.EventDispatchAction is a subclass of DispatchAction.

→ Its purpose is of that of its parent. i.e EventDispatchAction is used to create multi action controllers in a Struts application.

→ It supports this [illegible] for different use case if required.

→ EventDispatchAction usage involves the following steps

Step1: develop a user defined action class that extends EventDispatchAction.

Step2: for each use-case develop a user defined method with execute prototype but different method name.

Step3: Configure the action class (any number of times) with parameter attribute whose value is the user defined method name.

for eg

```
<action path="/emp"
Parameter="appointEmployee, removeEmployee"
    type="com.nareshit.action.EmployeeAction"
    name="employeeform">
```

OR

```
<action path="/Storeemp" Parameter="appointEmployee"
    type="com.nareshit.action.EmployeeAction"
    name="employeeform1" />
<action path="/deleteemp" Parameter="fireEmployee"
    type="com.nareshit.action.EmployeeAction"
    name="employeeform2" />
```

Step4 Submit button caption (request parameter value) can be any thing. But submit button name (request parameter) name should be the specified value in <action> tag i.e user defined method name or alias name.

for eg

```
<html:submit property="remove"><bean:message key="dlt" />
</html:submit>
```

Q: Develop a Struts Application in which, EventDispatchAction is used to ~~specify~~ unify multiple use-cases into one action class.

~~Assignment~~ from Mapping & lookupdispatch

Q: Explain about Modularicing a Struts application.

→ Dividing a mastere Struts application into decreet modulee is nothing but modulariciing the Struts application.

→ Each Module hae ite own jsps, formbeans, action Classes and configuration file.

⇒ To support projeet Modulee, modulariśation Concept ie given i Struts. It provides development Convenience.

→ Modulariciing a Struts application involves the following

Step1: Develop a configuration file pere module

for eg
- Struts-config.xml (default module)
- Struts-config-user.xml (user module)
- Struts-Config-admin.xml (admin module)

→ Configure the jsps, formbeans, action Classes of each module in ea configuration file.

Step2: Supply Configuration files to ActionServlet as Init Paramet

for eg

```
<Servlet>
<Servlet-name>frontController</Servlet-name>
<Servlet-Class> org.apache.Struts.action.ActionServlet </Servlet-
<init-param>
    <param-name> Config </param-name>
    <Param-value> /WEB-INF/Struts-config.xml </param-value>
</init-param>
<init-param>
    <param-name>Config/admin </param-name>
    <param-value>/WEB-INF/Struts-config-admin.xml </param
</init-param>
```

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

13/12/13

Q: How does Modularisation work in the background ?

→ As soon as the application is deployed, within the init method of ActionServlet, each configuration file is converted into ModuleConfig object and is stored into application scope. Each ModuleConfig object is stored in ServletContext with module prefix (appended to some constant used by framework) as attribute name.

→ Whenever client request comes, based on the module prefix, appropriate ModuleConfig object is retrieved from application scope to provide support for the use-case functionality.

Q: Explain about SwitchAction

→ It is one of the built in Struts action classes.

→ It is similare to that of ForwardAction except linking to the jsp of other module.

→ SwitchAction takes the module prefix from the query string of the link and switches the module before switching the control to the target page.

→ following steps are involved in using SwitchAction in a Struts application.

Step 1 Configure this built-in action class in struts-config.xml

- <action path = "/switchmodule"
type = "org.apache.struts.actions.SwitchAction" />

Step2 Create the hyperlink as follows

<A HREF = "./switchmodule.do?Prefix=/admin&Page=/some.do">
CLICK HERE </A>

Q: Explain about double submit issue in a Struts application.

→ Users may submit the same webform twich which is nothing but double submission issue.

→ Double submission of the web form some times irritating for the end-users and some times has serious consequences.

→ Struts developers should address this issue. Even though the form is double submitted, we should ensure that there is no side effect on the application (application's data).

Q: How to prevent the dangers caused by the double submission of the form in a Struts application?

→ Org.apache.struts.action.Action class has 3 methods to deal with double submission of the web form.

1. saveToken(request):- This method generates a unique number called a token and stores into Session scope.

2. resetToken(request): This method destroys the token stored in the session object.

3. isTokenValid(request): This method captures the incoming token from the client (as hidden form field) and compares it with already stored token in session scope. If it matches, it returns true otherwise false.

→ <html:form> tag is responsible to retrieve the token from session scope and converts into a hidden form field

Q: Develop a Struts application in which double Submission issue is addressed ?

doublesubmitapplication
- index.jsp
- register.jsp
- Success.jsp
- Warning.jsp
- WEB-INF
  - Web.xml
  - Struts-config.xml
  - Src
    - RegisterAction.java
    - RegisterForm.java
  - Classes
    - *.class
  - lib
    - *.jar(6+1)

http://localhost:8081/doublesubmitapplication.

RegisterAction.java

```
Public class RegisterAction extends DispatchAction
{ Public ActionForward registerPage(---)
  { SaveToken (request); // Save token before showing
    return register.jsp
  return mapping.findForward("register");
}
Public ActionForward register(----)
{ String responsekey = "Success";
  if (this.isTokenValid(request))
  { resetToken(request);
    // invoke Model Component for registration
  }
else
{
```

```
            responsekey = "warning";
        }
        return mapping.findForward(responsekey);
    }
}
```

Struts-Config.xml

```
<struts-config>
  <form-beans>
    <form-bean name="registerForm"
      type="com.nareshit.form.RegisterForm" />
  </form-beans>
  <action-mappings>
    <action path="/registerAction" name="registerForm"
      type="com.nareshit.Controller.RegisterAction"
        parameter="usr" >
      <forward name="register" path="/register.jsp" />
      <forward name="success" path="/success.jsp" />
      <forward name="warning" path="/warning.jsp" />
    </action>
  </action-mappings>
</struts-config>
```

Index.jsp

```
<jsp:forward page="registerAction.do?usr=registerPage" />
```

register.jsp

```
<%@ taglib uri = "http://Strouts.apache.org/tags-html" prefix="html"%>
<html>
  <body bgcolor = "cyan">
  <center>
     <h1> Registration Screen </h1>
   <html:form action="/registerAction">
  Name <html:text property="name"/> <BR><BR>
  <html:submit property="usre"> register </html:submit>
  </html:form>
  </center>
   </body>
    </html>
```

Success.jsp

```
   <html>
   <body bgcolor="cyan">
   <center>
      <H1> Registration Successful </H1>
    </center>
     </body>
      </html>
```

Warning.jsp

```
   <html>
     <body bgcolor="cyan">
     <H1> Your Form is already Submitted. Don't Submit again </H1>
     </body>
      </html>
```

16.12.13

Q: Develop a Java web application (Struts application) in which doun
use-case is implemented ?

downloadapplication
  download.jsp
  downloadapp.txt (user download this file)
  WEB-INF
    web.xml
    Struts-config.xml
    Src
      FileDownloadAction.java
    Classes
      *.Class
      lib
        *.jar (5 jar files)

http://localhost:8081/downloadapplication

Struts-config.xml

```
<Struts-config>
  <action-mappings>
    <action path="/downloadfile"
      type="com.nareshit.action.FileDownloadAction" />
  </action-mappings>
</Struts-config>
```

download.jsp

```
<%@ taglib prefix="html" uri="http://struts.apache.org/tags-l
<html>
  <body bgcolor="cyan">
  <h1>
    <html:link action="/downloadfile">downloadhere</html:lin
  </h1>
  </body>
</html>
```

Web.xml

Make download.jsp the home page.

FileDownloadAction.java

```
Package com.nareshit.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import java.io.*;
Public Class fileDownloadAction extends Action
{ Public ActionForward execute(ActionMapping mapping,
ActionForm form, HttpServletRequest request, HttpServletResponse
response) throws Exception
{
    response.setContentType("text/plain");
    response.setHeader("Content-disposition","attachment;
    filename=downloadapp.txt");
String path = getServlet().getServletContext().getRealPath("")+
"/"+"download/app.txt";
FileInputStream fis = new FileInputStream(Path);
BufferedInputStream bis = new BufferedInputStream(fis);
byte[] bytes = new byte[bis.available()];
OutputStream ops = response.getOutputStream(); // browsers stream
    bis.read(bytes);
    ops.write(bytes);
    bis.close();
    ops.close();
    return null;
}
}
```

NOTE: If we use org.apache.struts.actions.DownloadAction as Supere class for the userdefined action class, IO code for file reading and file writing is provided by framework.

FileDownloadAction.java

```
package com.nareshit.action;
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.apache.struts.actions.*;
Public Class FileDownloadAction extends DownloadAction
{
public StreamInfo getStreamInfo(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) throws Exception
{ response.setHeader("Content-disposition", "attachment; filename=downloadapp.txt");
Returen new ResourceStreamInfo("text/plain", getServlet().getServletContext(), "downloadapp.txt");
}
}
```

Q: Explain about IncludeAction.

→ org.apache.struts.actions.IncludeAction is used to include so .jsp into another jsp in logical name based manner.

→ The following steps are involved in using IncludeAction

Step1: `<action path="/footere" type="org.apache.struts.actions.IncludeAction" Parameter="/footere.jsp" />`

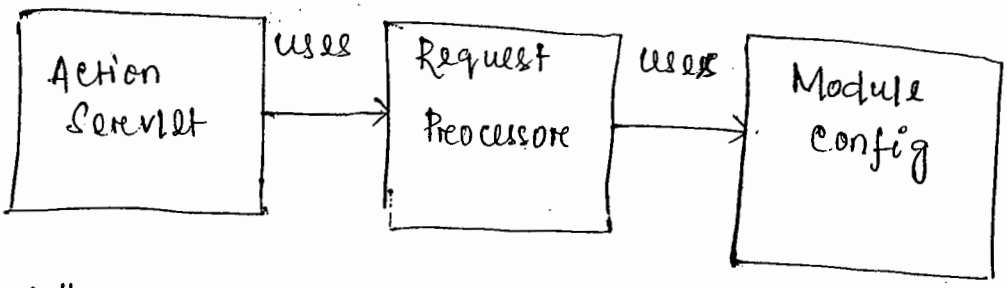
struts. In include jsp file we have following entry

<jsp:include page="/footer.do" />

Note: This built-in action is seldom used. (Seldom - almost not used)

17/12/13

Q: How does Struts work in the background?



→ Whenever client request comes, ActionServlet invokes RequestProcessor to implement the use-case required framework functionality. RequestProcessor takes the configuration information required for the use-case from moduleConfig object.

Q: What is the general structure of ActionServlet?

```
Public Class ActionServlet extends HttpServlet
  {
    Public void init(ServletConfig config)
    {
    /*
    1. reading configuration file
    2. Converting xml into ModuleConfig object using digester
       API and storing the same into application scope.
    3. instatiating and initializing plug-ins if any
    */
  }
Public Void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException
    {
```

process(request, response);

}

Private void process(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

ModuleConfig mconfig = getModuleConfig();

RequestProcessor rep = getRequestProcessorForModule(mconfig);

rep.Process(request, response);

{

}

Q: Explain about ModuleConfig.

→ Object Oriented representation of Struts configuration file is nothing but ModuleConfig object.

→ Each tag of Struts configuration file is converted into a Java object and the references of all those Java objects are stored into ModuleConfig object.

→ Framework (RequestProcessor) uses ModuleConfig object to deal with the MVC functionality of the application.

→ ModuleConfig object is the meta information repository of the Struts application.

Q: Explain about RequestProcessor.

→ RequestProcessor is the work horse of Struts for each Client req

→ RequestProcessor is a normal Java class. It is not a web component class.

→ for each use-case of the Struts application, framework functionality is performed by RP.

→ Requestprocessor does

a. formbean instantiation
b. populating the bean fields.
c. invoking the validation logic
d. Creating action class instance.
e. & invoking execute method etc.

→ RP has methods to do each task. These methods are processxxx. method.

for e.g.

a. ProcessActionForm()
b. ProcessPopulate()
c. processValidate()
d. processActionCreate()
e. processActionPerform() etc.

Q: How to have user defined Requestprocessor for a Struts application?

→ for almost all the requirements built-in RP is sufficient. Very rarely to interfere in framework's functionality we need to have our own RP.

Step1: Develop a Java Class that extends RP

Step2: Overeride required method of RP whose functionality we want to modify.

Step3: Specified the user defined RP in Configuration file using <Controller> tag.

Q: Explain about a plug-in in a Struts application.

→ A Java class that implements org.apache.struts.action.PlugIn Interface is nothing but a plug-in in a Struts application.

→ Whatever resources required to the Struts application at the run time of its deployment can be provided through plugin class.

→ plugin is instantiated with ActionServlet and destroyed wit

→ There are two kinds of plug-ins
1. built-in plugins (validator and tiles)
2. User defined plug in

→ following steps are involved in using a user-defined plugin in Struts Application.

Step1: Develop a Java class that implements PlugIn interface.

Step2: Override init() & destroy() methods of PlugIn interf

Step3: If we want to supply any values through Struts-config.x class and define one setter method.

Step4: Configure the userdefined PlugIn class in Struts configuration file using <plug-in> tag.

18.12.17

Q: Develop a Struts application in which, userdefined request processor and user defined plugin are used.

pluginrequestprocessorapplication
- failure.jsp
- home.jsp
- welcome.jsp
- WEB-INF
  - web.xml
  - Struts-config.xml
- Src
  - CounterPlugIn.java
  - FormalityAction.java
  - MyRequestProcessor.java
- Classes
  - *.class
- lib
  - *.jar (5 jar files)

http://localhost:8081/pluginrequestprocessorapplication.

MyRequestProcessor.java....

```
Package com.nareshit.struts;
import javax.servlet.http.*;
import javax.servlet.*;
import org.apache.struts.action.RequestProcessor;
import java.io.*;
Public class MyRequestProcessor extends RequestProcessor
{
Public boolean processPreprocess(HttpServletRequest request, Http-
ServletResponse response)
    {
    String method = request.getMethod();
    if (method.equalsIgnoreCase("post"))
        return true;
RequestDispatcher rd = request.getRequestDispatcher("failure.jsp");
try{
    rd.forward(request, response);
    }
catch(Exception e){ }
        return false;
    }
}
```

Note: → ProcessPreprocess() is the entry point into the Struts functionality. This method returns boolean. By default, RequestProcessor implemented this method to return true.

→ We can override this method and write any code that has to be executed before any use case functionality is executed. for eg. Security logic.

→ If this method returns false, no more processing of the request further.

CountereplugIn.java

```
Package com.nareshit.struts;
import org.apache.struts.config.ModuleConfig;
import org.apache.struts.action.PlugIn;
import org.apache.struts.action.ActionServlet;
public class CountereplugIn implements PlugIn
{ private int initialcount;
public void setInitialCount(int i)
{ s.o.p("setHere called with value:"+i);
initialcount=i;
}
public void init(ActionServlet servlet, ModuleConfig config)
{ s.o.p("init called");
servlet.getServletContext().setAttribute("counter.", initialCo
}
public void destroy(){ }
}
```

Struts-config.xml

```
<Struts-Config>
<action-mappings>
<action path="/hello" type="com.nareshit.struts.FormalityActio
<forward name="success" path="/welcome.jsp" />
</action>
</action-mappings>
<Controller
ProcessorClass="com.nareshit.struts.MyRequestProcessor" />
<message-resources
Parameter="ApplicationResources" />
<Plug-in className="com.nareshit.struts.CounterPlugIn" >
<set-Property property="initialcount" value="1000" />
</Plug-in>
</Struts-Config>
```

Home.jsp

```
<Html>
  <Body bgcolore="yellow">
  <centre>
    <H1> Click here for get request </H1>
    <form action="hello.do">
   <input type="submit" value="GET REQUEST"/>
  </form>
    <H1> Click here for post request </H1>
   <form action="hello.do" method="post">
   <input type="submit" value="POST REQUEST"/>
  </form>
   <center>
   </html>
```

failure.jsp

```
<html>
  <Body bgcolor="gray">
    <H1> Next time don't click on get button </H1>
  </Body>
   </html>
```

Welcome.jsp

```
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean"
<Html>
  <Body bgcolor="green">
  <H1> Welcome to www.nareshit.com </H1>
  <H2> Number of times this page visited:
      <bean:write name="counter"/> </H2>
   </body>
   </html>
```

FormalityAction.java

```
public class FormalityAction extends Action
{ public ActionForward execute (AM mapping, AF af,
  HSR request, HSR response) throws Exception
  { return am.findForward ("success");
  }
}
```

Struts 2.X

19.12.13

Q: What are the similarities between Struts 1.x and Struts 2.x ?

→ Both are MVC based Java web application frameworks from

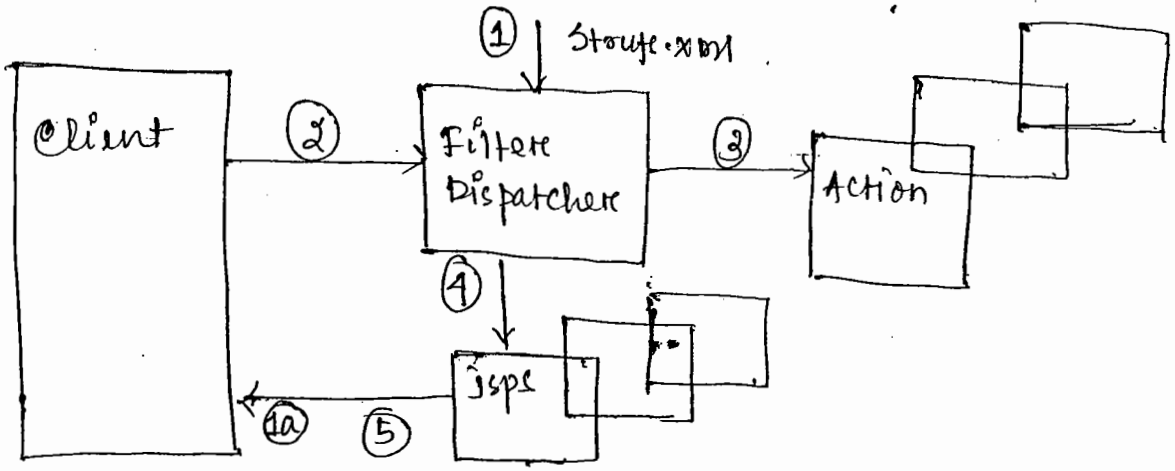
→ Both are action based use-case implementation frameworks.

Q: What are the differences between Struts 1.x and Struts 2.x ?

Struts 1.x	Struts 2.x
(1). ActionServlet is the front-Controller	(1) filterDispatcher which is a Servl filter.
(2). RequestProcessor is the backbone of the framework.	(2) Interceptors replaced RequestProc
(3). form beans are the data containers of user input.	(3). no form beans. action class instance acts as the data container for user-input.
(4). action class is a singleton.	(4) not a singleton.
(5). action class has thread safety issue to deal with.	(5). no thread-safety issue.
(6). action class is not a POJO.	(6) It can be a POJO.
(7) action class has Servlet dependencies	(7) no Servlet dependencies

(8) Struts-config.xml is the configuration file.	(8). Struts.xml
(9) ~~Interface~~ Class based framework	(9). Interface based framework.
(10) heavy weight	(10). light weight
(11) annotations support not at all there.	(11). to some extent achieved.

Q: What is the architecture of Struts 2.x ?



→ Org.apache.struts2.dispatcher.FilterDispatcher is the front-controller in a Struts 2 application.

→ Application deployment time, FilterDispatcher reads Struts.xml and understands the flow control of all the use-cases of the application.

→ When client request comes, FilterDispatcher receives the control and invokes the action.

→ Based on the view mapping information from the execute() method of user defined action instance, FilterDispatcher switches the control to the appropriate view component that produces the response for the client.

Q: Develop a Struts 2 application that implements the use-case of user authentication.

loginapplication
- login.jsp
- welcome.jsp
- loginfailed.jsp

WEB-INF
  web.xml
src
  LoginAction.java
  LoginModel.java
Classes
  Struts.xml
+ v. class
lib
  commons-logging-1.1.1.jar
  freemarker-2.3.8.jar
  ognl-2.6.11.jar
  Struts2-core-2.0.6.jar
  xwork-2.0.1.jar

http://localhost:8081/loginapplication.

20.12.13

LoginAction.java

```
Package com.nareshit.action;
import com.nareshit.model.LoginModel;
Public Class LoginAction
  { Private String username;
    Private String Password;
    // 2 setters & 2 getters
  Public string execute()
    { boolean flag = lm.isAuthenticated(username, Password);
      if (flag)
        return "ok";
        return "notok";
    }
}
```

→ LoginModel lm = new LoginModel();

· login.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<HTML>
  <BODY BGCOLOR="cyan">
  <center>
    <H1> Login Screen </H1>
  <S:form action="login" method="post">
  <S:textfield label="USERNAME" name="username" /> <BR>
  <S: password label="PASSWORD" name="password" /> <BR>
  <S: submit value="login" />
  </S:form>
  </center>
  </BODY>
  </HTML>
```

Note: welcome.jsp, loginfailed.jsp and loginModel.java from Struts 1.x "loginapplication".

Web.xml:

```
<web-app>
<filter>
<filter-name>frontcontroller</filter-name>
<filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-cla
</filter>
<filter-mapping>
  <filter-name>frontcontroller</filter-name>
  <url-pattern>/*</url-Pattern>
</filter-mapping>
<welcome-file-list>
  <welcome-file>login.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Struts.xml

```
<Struts>
  <package name="loginpack" extends="Struts-default">
  <action name="login" Class="com.nareshit.action.LoginAction">
   <result name="ok">/welcome.jsp</result>
   <result name="notok">/loginfailed.jsp</result>
  </action>
 </package>
</Struts>
```

Q: Modify Struts1.x "registrationapplication" into that of Struts2.

```
registrationapplication
    register.jsp
    Sucess.jsp
    registrationfailed.jsp
      WEB-INF
        web.xml
        src
          RegisterAction.java
          RegisterModel.java
            DBUtil.java
        Classes
          Struts.xml
          *.class
            lib
              ojdbc14.jar
              *.jar (5 jar files)
```

http://localhost:8081/registrationapplication

RegisterAction.java

```
Package com.nareshit.action;
import com.nareshit.model.RegisterModel;
Public class RegisterAction
    {  RegisterModel rm = new RegisterModel();
```

```
Private String username;
Private String Password;
Private String emailid;
// 3 setters & getters
Public String execute()
{ boolean flag = rem.registerUser(username, password, emailid);
  if (flag)
      return "ok";
      return "notok";
  }
}
```

Web.xml

→ Make register.jsp the home page.

Struts.xml:

```
<Struts>
  <Package name="registerpack" extends="struts-default">
  <action name="register"
      class="com.nareshit.action.RegisterAction">
  <result name="ok">/success.jsp</result>
  <result name="notok">/registrationfailed.jsp</result>
  </action>
  </Package>
  </Struts>
```

register.jsp

```
<%@ taglib prefix="s" uri="/struts-tags" %>
<Html>
<Body Bgcolor="cyan">
<center>
<H1> Registration Screen </H1>
```

```
<s:form action="register" method="post">
<s:textfield label="USERNAME" name="username" /><BR>
<s:password label="PASSWORD" name="password" /><BR>
<s:textfield label="EMAILID" name="emailid" /><BR>
<s:submit value="register" />
</s:form>
</center>
</body>
</html>
```

Note:
Other two jsps and two java files from Struts1.x applicat