JavaServer Faces Interview Questions

1) What is JSF?

JSF stands for Java Server Faces. JSF has set of pre-assembled User Interface (UI). By this it means complex components are pre-coded and can be used with ease. It is event-driven programming model. By that it means that JSF has all necessary code for event handling and component organization. Application programmers can concentrate on application logic rather sending effort on these issues. It has component model that enables third-party components to be added like AJAX.

2) What is required for JSF to get started?

Following things required for JSF:

- JDK (Java SE Development Kit)
- JSF 1.2
- Application Server (Tomcat or any standard application server)
- Integrated Development Environment (IDE) Ex. Netbeans 5.5, Eclipse 3.2.x, etc.

Once JDK and Application Server is downloaded and configured, one can copy the JSF jar files to JSF project and could just start coding. :-)
If IDE is used, it will make things very smooth and will save your time.

3) What is JSF architecture?

JSF was developed using MVC (a.k.a Model View Controller) design pattern so that applications can be scaled better with greater maintainability. It is driven by Java Community Process (JCP) and has become a standard. The advantage of JSF is that itâ€™s both a Java Web user â€“ interface and a framework that fits well with the MVC. It provides clean separation between presentation and behavior. UI (a.k.a User Interface) can be created by page author using reusable UI components and business logic part can be implemented using managed beans.

4) How JSF different from conventional JSP / Servlet Model?

JSF much more plumbing that JSP developers have to implement by hand, such as page navigation and validation. One can think of JSP and servlets as the â€œassembly languageâ€? under the hood of the high-level JSF framework.

5) How the components of JSF are rendered? An Example

In an application add the JSF libraries. Further in the .jsp page one has to add the tag library like:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
```

Or one can try XML style as well:

```
<?xml version="1.0"?>
<jsp:root version="2.0" xmlns:jsp="http://java.sun.com/JSP/Page"
                        xmlns:f="http://java.sun.com/jsf/core"
                        xmlns:h="http://java.sun.com/jsf/html">
```

Once this is done, one can access the JSF components using the prefix attached. If working with an IDE (a.k.a Integrated Development Environment) one can easily add JSF but when working without them one also has to update/make the faces-config.xml and have to populate the file with classes i.e. Managed Beans between

```
<faces-config> </faces-config> tags
```

6) How to declare the Navigation Rules for JSF?

Navigation rules tells JSF implementation which page to send back to the browser after a form has been submitted. For ex. for a login page, after the login gets successful, it should go to Main page, else to return on the same login page, for that we have to code as:

```
<navigation-rule>
   <from-view-id>/login.jsp</from-view-id>
   <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>/main.jsp<to-view-id>
   </navigation-case>
   <navigation-case>
        <from-outcome>fail</from-outcome>
        <to-view-id>/login.jsp<to-view-id>
   </navigation-case>
</navigation-rule>
```

from-outcome to be match with action attribute of the command button of the login.jsp as:

```
<h:commandbutton value="Login" action="login"/>
```

Secondly, it should also match with the navigation rule in face-config.xml as

```
<managed-bean>
   <managed-bean-name>user</managed-bean-name>
   <managed-bean-class>core.jsf.LoginBean</managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

In the UI component, to be declared / used as:

```
<h:inputText value="#{user.name}"/>
```

value attribute refers to name property of the user bean.

7) How do I configure the configuration file?

The configuration file used is our old web.xml, if we use some IDE it will be pretty simple to generate but the contents will be something like below:

```
<?xml version=&quote;1.0&quote; encoding=&quote;UTF-8&quote;?>

<web-app version=&quote;2.4&quote; xmlns=&quote;http://java.sun.com/xml/ns/j2ee&qu
        xmlns:xsi=&quote;http://www.w3.org/2001/XMLSchema-instance&quote;
        xsi:schemaLocation=&quote;http://java.sun.com/xml/ns/j2ee http://java.sun.
   <context-param>
        <param-name>com.sun.faces.verifyObjects</param-name>
        <param-value>false</param-value>
   </context-param>

   <context-param>
        <param-name>com.sun.faces.validateXml</param-name>
        <param-value>true</param-value>
   </context-param>

   <context-param>
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
   </context-param>

   <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>

   <session-config>
        <session-timeout>
           30
        </session-timeout>
   </session-config>

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

The unique thing about this file is ?servlet mapping?. JSF pages are processed by a servlet known to be part of JSF implementation code. In the example above, it has extension of .faces. It would be wrong to point your browser to http://localhost:8080/MyJSF/login.jsp,

but it has to be http://localhost:8080/MyJSF/login.faces. If you want that your pages to be with .jsf, it can be done with small modification :-),

```
<servlet-mapping>
<servlet-name>Faces Servlet</servlet-name>
<url-pattern>*.jsf</url-pattern>

<servlet-mapping>
```

8) What is JSF framework?

JSF framework can be explained with the following diagram:

As can be seen in Figure 1, JSF interacts with Client Devices which ties together with presentation, navigation and event handling and business logic of web tier model. Hence JSF is limited to presentation logic / tier. For Database tier i.e. Database and Web services one has to rely on other services.

9) How does JSF depict the MVC (a.k.a Model View Controller) model?

The data that is manipulated in form or the other is done by model. The data presented to user in one form or the other is done by view. JSF is connects the view and the model. View can be depicted as shown by:

```
<h:inputText value="#{user.name}"/>
```

JSF acts as controller by way of action processing done by the user or triggering of an event. For ex.

```
<h:commandbutton value="Login" action="login"/>
```

, this button event will triggered by the user on Button press, which will invoke the login Bean as stated in the faces-config.xml file. Hence, it could be summarized as below: User Button Click -> form submission to server -> invocation of Bean class -> result thrown by Bean class caught be navigation rule -> navigation rule based on action directs to specific page.

10) What does it mean by rendering of page in JSF?

Every JSF page as described has various components made with the help of JSF library. JSF may contain h:form, h:inputText, h:commandButton, etc. Each of these are rendered (translated) to HTML output. This process is called encoding. The encoding procedure also assigns each component with a unique ID assigned by framework. The ID generated is random.

11) What is JavaServer Faces?

JavaServer Faces (JSF) is a user interface (UI) framework for Java web applications. It is designed to significantly ease the burden of writing and maintaining applications that run on a Java application server and render their UIs back to a target client. JSF provides ease-of-use in the following ways:

- Makes it easy to construct a UI from a set of reusable UI components
- Simplifies migration of application data to and from the UI
- Helps manage UI state across server requests
- Provides a simple model for wiring client-generated events to server-side application code
- Allows custom UI components to be easily built and re-used

Most importantly, JSF establishes standards which are designed to be leveraged by tools to provide a developer experience which is accessible to a wide variety of developer types, ranging from corporate developers to systems programmers. A "corporate developer" is characterized as an individual who is proficient in writing procedural code and business logic, but is not necessarily skilled in object-oriented programming. A "systems programmer" understands object-oriented fundamentals, including abstraction and designing for re-use. A corporate developer typically relies on tools for development, while a system programmer may define his or her tool as a text editor for writing code. Therefore, JSF is designed to be tooled, but also exposes the framework and programming model as APIs so that it can be used outside of tools, as is sometimes required by systems programmers.

12) How to pass a parameter to the JSF application using the URL string?

if you have the following URL: http://your_server/your_app/product.jsf?id=777, you access the passing parameter id with the following lines of java code:

```
FacesContext fc = FacesContext.getCurrentInstance();
String id = (String) fc.getExternalContext().getRequestParameterMap().get("id");
```

From the page, you can access the same parameter using the predefined variable with name param. For example,

```
<h:outputText value="#{param['id']}" />
```

Note: You have to call the jsf page directly and using the servlet mapping.

13) How to add context path to URL for outputLink?

Current JSF implementation does not add the context path for outputLink if the defined path starts with '/'. To correct this problem use #{facesContext.externalContext.requestContextPath} prefix at the beginning of the outputLink value attribute. For example:

```
<h:outputLink value="#{facesContext.externalContext.requestContextPath}/myPage.fac
```

14) How to get current page URL from backing bean?

You can get a reference to the HTTP request object via FacesContext like this:

```
FacesContext fc = FacesContext.getCurrentInstance();
HttpServletRequest request = (HttpServletRequest) fc.getExternalContext().getReque
```

and then use the normal request methods to obtain path information. Alternatively,

```
context.getViewRoot().getViewId();
```

will return you the name of the current JSP (JSF view IDs are basically just JSP path names).

15) How to access web.xml init parameters from java code?

You can get it using externalContext getInitParameter method. For example, if you have:

```
<context-param>
   <param-name>connectionString</param-name>
   <param-value>jdbc:oracle:thin:scott/tiger@cartman:1521:O901DB</param-value>
</context-param>
```

You can access this connection string with:

```
FacesContext fc = FacesContext.getCurrentInstance();
String connection = fc.getExternalContext().getInitParameter("connectionString");
```

16) How to access web.xml init parameters from jsp page?

You can get it using initParam pre-defined JSF EL valiable.
For example, if you have:

```
<context-param>
 <param-name>productId</param-name>
 <param-value>2004Q4</param-value>
</context-param>
```

You can access this parameter with #{initParam['productId']} . For example:

```
Product Id: <h:outputText value="#{initParam['productId']}"/>
```

17) How to terminate the session?

In order to terminate the session you can use session invalidate method.
This is an example how to terminate the session from the action method of a backing bean:

```
public String logout() {
  FacesContext fc = FacesContext.getCurrentInstance();
```

```
  HttpSession session = (HttpSession) fc.getExternalContext().getSession(false);
  session.invalidate();
  return "login_page";
}
```

The following code snippet allows to terminate the session from the jsp page:

```
<% session.invalidate(); %> <c:redirect url="loginPage.jsf" />
```

18) How to implement "Please, Wait..." page?

The client-side solution might be very simple. You can wrap the jsp page (or part of it you want to hide) into the DIV, then you can add one more DIV that appears when user clicks the submit button. This DIV can contain the animated gif you speak about or any other content.
Scenario: when user clicks the button, the JavaScript function is called. This function hides the page and shows the "Wait" DIV. You can customize the look-n-fill with CSS if you like. This is a working example:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<f:loadBundle basename="demo.bundle.Messages" var="Message"/>

<html>
<head>
   <title>Input Name Page</title>
   <script>
     function gowait() {
        document.getElementById("main").style.visibility="hidden";
        document.getElementById("wait").style.visibility="visible";
     }
   </script>
</head>

<body bgcolor="white">
<f:view>
   <div id="main">
       <h1><h:outputText value="#{Message.inputname_header}"/></h1>
       <h:messages style="color: red"/>
       <h:form id="helloForm">
         <h:outputText value="#{Message.prompt}"/>
         <h:inputText id="userName" value="#{GetNameBean.userName}" required="true
           <f:validateLength minimum="2" maximum="20"/>
         </h:inputText>
         <h:commandButton onclick="gowait()" id="submit" action="#{GetNameBean.act
       </h:form>
   </div>
   <div id="wait" style="visibility:hidden; position: absolute; top: 0; left: 0">
       <table width="100%" height ="300px">
         <tr>
           <td align="center" valign="middle">
             <h2>Please, wait...</h2>
           </td>
```

```
          </tr>
        </table>
    </div>
</f:view>
</body>
</html>
```

If you want to have an animated gif of the "Wait" Page, the gif should be reloaded after the form is just submitted. So, assign the id for your image and then add reload code that will be called after some short delay. For the example above, it might be:

```
<script>
 function gowait() {
   document.getElementById("main").style.visibility="hidden";
   document.getElementById("wait").style.visibility="visible";
   window.setTimeout('showProgress()', 500);
 }
  function showProgress(){
   var wg = document.getElementById("waitgif");
   wg.src=wg.src;
 }
</script>
....
....
....

<img id="waitgif" src="animated.gif">
```

19) How to reload the page after ValueChangeListener is invoked?

At the end of the ValueChangeListener, call
FacesContext.getCurrentInstance().renderResponse()

20) How to download PDF file with JSF?

This is an code example how it can be done with action listener of the backing bean.
Add the following method to the backing bean:

```
public void viewPdf(ActionEvent event) {
   String filename = "filename.pdf";

   // use your own method that reads file to the byte array
   byte[] pdf = getTheContentOfTheFile(filename);

   FacesContext faces = FacesContext.getCurrentInstance();
   HttpServletResponse response = (HttpServletResponse) faces.getExternalContext()
   response.setContentType("application/pdf");
   response.setContentLength(pdf.length);
   response.setHeader( "Content-disposition", "inline; filename=\""+fileName+"\"")
   try {
```

```
            ServletOutputStream out;
            out = response.getOutputStream();
            out.write(pdf);
        } catch (IOException e) {
            e.printStackTrace();
        }
        faces.responseComplete();
    }
```

This is a jsp file snippet:

```
<h:commandButton immediate="true" actionListener="#{backingBean.viewPdf}" value="R
```

21) How to show Confirmation Dialog when user Click the Command Link?

h:commandLink assign the onclick attribute for internal use. So, you cannot use it to write your own code. This problem will fixed in the JSF 1.2. For the current JSF version you can use onmousedown event that occurs before onclick. <script language="javascript"> function ConfirmDelete(link) { var delete = confirm('Do you want to Delete?'); if (delete == true) { link.onclick(); } } </script> . . . . <h:commandLink action="delete" onmousedown="return ConfirmDelete(this);"> <h:outputText value="delete it"/> </h:commandLink>

22) What is the different between getRequestParameterMap() and getRequestParameterValuesMap()

getRequestParameterValuesMap() similar to getRequestParameterMap(), but contains multiple values for for the parameters with the same name. It is important if you one of the components such as <h:selectMany>.

23) Is it possible to have more than one Faces Configuration file?

Yes. You can define the list of the configuration files in the web.xml.
This is an example:

```
<context-param>
   <param-name>javax.faces.CONFIG_FILES</param-name>
   <param-value>/WEB-INF/faces-config-navigation.xml,/WEB-INF/faces-beans.xml</par
</context-param>
```

Note: Do not register /WEB-INF/faces-config.xml file in the web.xml . Otherwise, the JSF implementation will process it twice.
Hi there, I guess the Note: column should have been meant or intended for "faces-config.xml" file as thats the default configuration file for JSF (which is similar to struts-config.xml for Struts!!). faces-context.xml file sounds like the user defined config file similar to the aforementioned two xml files.

24) How to mask actual URL to the JSF page?

You'll need to implement your own version of javax.faces.ViewHandler which does what you need. Then, you register your own view handler in faces-config.xml.
Here's a simple abstract ViewHandler you can extend and then implement the 3 abstract methods for. The abstract methods you override here are where you'll do your conversions to/from URI to physical paths on the file system. This information is just passed right along to the default ViewHandler for JSF to deal with in the usual way. For example, you could override these methods to add and remove the file extension of an incoming view id (like in your example), for extension-less view URIs.

```
import java.io.IOException;
import java.util.Locale;

import javax.faces.FacesException;
import javax.faces.application.ViewHandler;
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * A facade view handler which maps URIs into actual physical views that the
 * underlying implementation can deal with regularly.
 * Therefore, all internal references to view ids, for example in faces-config,
 * will use the path to the physical files. Everything publicized, however, will
 * see a "converted" / facade url.
*/
public abstract class SimpleConverterViewHandler extends ViewHandler {
   private static final Log LOG = LogFactory.getLog(SimpleConverterViewHandler.cla

   private final ViewHandler base;

   public SimpleConverterViewHandler(ViewHandler base) {
     this.base = base;
   }

   /**
     * Distinguishes a URI from a physical file view.
     * Tests if a view id is in the expected format -- the format corresponding
     * to the physical file views, as opposed to the URIs.
     * This test is necessary because JSF takes the view ID from the
     * faces-config navigation, and calls renderView() on it, etc.
   */
   public abstract boolean isViewFormat(FacesContext context, String viewId);

   /**
     * Convert a private file path (view id) into a public URI.
   */
   public abstract String convertViewToURI(FacesContext context, String viewId);

   /**
```

```
  * Convert a public URI into a private file path (view id)
  * note: uri always starts with "/";
*/
public abstract String convertURIToView(FacesContext context, String uri);

public String getActionURL(FacesContext context, String viewId) {
  // NOTE: this is used for FORM actions.

  String newViewId = convertViewToURI(context, viewId);
  LOG.debug("getViewIdPath: " + viewId + "->" + newViewId);
  return base.getActionURL(context, newViewId);
}

private String doConvertURIToView(FacesContext context, String requestURI) {
  if (isViewFormat(context, requestURI)) {
    return requestURI;
  } else {
    return convertURIToView(context, requestURI);
  }
}

public void renderView(FacesContext context, UIViewRoot viewToRender)
  throws IOException, FacesException {
  if (null == context || null == viewToRender)
    throw new NullPointerException("null context or view");

    String requestURI = viewToRender.getViewId();
    String newViewId = doConvertURIToView(context, requestURI);
    LOG.debug("renderView: " + requestURI + "->" + newViewId);
    viewToRender.setViewId(newViewId);

    base.renderView(context, viewToRender);
  }

public UIViewRoot restoreView(FacesContext context, String viewId) {
  String newViewId = doConvertURIToView(context, viewId);
  LOG.debug("restoreView: " + viewId + "->" + newViewId);
  return base.restoreView(context, newViewId);
}

public Locale calculateLocale(FacesContext arg0) {
  return base.calculateLocale(arg0);
}

public String calculateRenderKitId(FacesContext arg0) {
  return base.calculateRenderKitId(arg0);
}

public UIViewRoot createView(FacesContext arg0, String arg1) {
  return base.createView(arg0, arg1);
}

public String getResourceURL(FacesContext arg0, String arg1) {
  return base.getResourceURL(arg0, arg1);
```

```
    }

    public void writeState(FacesContext arg0) throws IOException {
      base.writeState(arg0);
    }

}
```

25) How to print out html markup with h:outputText?

The h:outputText has attribute escape that allows to escape the html markup. By default, it equals to "true". It means all the special symbols will be replaced with '&' codes. If you set it to "false", the text will be printed out without ecsaping.

```
For example, <h:outputText value="<b>This is a text</b>"/>

will be printed out like:

<b>This is a text</b>

In case of <h:outputText escape="false" value="<b>This is a text</b>"/>

you will get:

This is a text
```

26) h:inputSecret field becomes empty when page is reloaded. How to fix this?

Set redisplay=true, it is false by default.