Contents

[JavaServer Faces (JSF) 1](#_Toc417049971)

[What is JSF? 1](#_Toc417049972)

[JSF Pages 1](#_Toc417049973)

[What JSF Looks Like 1](#_Toc417049974)

[Why JSF? 2](#_Toc417049975)

[Model-View Controller 2](#_Toc417049976)

[The JSF Life Cycle 2](#_Toc417049977)

[The Lifecycle Phases 2](#_Toc417049978)

[JSF Request Processing 3](#_Toc417049979)

[JSF Lifecycle 4](#_Toc417049980)

[How JSF Works 4](#_Toc417049981)

[Facelets 5](#_Toc417049982)

[Event Handling 5](#_Toc417049983)

[Application Scopes 5](#_Toc417049984)

[How Scopes Work 5](#_Toc417049985)

[Application 5](#_Toc417049986)

[Session 5](#_Toc417049987)

[Request 5](#_Toc417049988)

[View 6](#_Toc417049989)

[Beans 6](#_Toc417049990)

[What a Bean Is 6](#_Toc417049991)

[Binding to Managed Beans 6](#_Toc417049992)

[SessionMap, ApplicationMap 6](#_Toc417049993)

[Using JavaScript in JSF 6](#_Toc417049994)

[JSF 2.x AJAX support 6](#_Toc417049995)

[JavaMail 6](#_Toc417049996)

[GET parameters 6](#_Toc417049997)

[JDBC, Data Sources 6](#_Toc417049998)

[loading driver 6](#_Toc417049999)

[getting a connection 6](#_Toc417050000)

[preparing SQL statements 6](#_Toc417050001)

[running the query/update 6](#_Toc417050002)

[getting the results 6](#_Toc417050003)

[Using a data source to access and commit changes to a database 6](#_Toc417050004)

[XML 7](#_Toc417050005)

[Why XML? 7](#_Toc417050006)

[Difference between HTML and XML 7](#_Toc417050007)

[Well-formedness vs. Validity 7](#_Toc417050008)

[Elements vs. Attributes 7](#_Toc417050009)

[XML Schema 7](#_Toc417050010)

[XML vs. JSON 7](#_Toc417050011)

[Web Services 7](#_Toc417050012)

[What is a Web Service? 7](#_Toc417050013)

[Why Use Web Services? 7](#_Toc417050014)

[Problems & Concerns 7](#_Toc417050015)

[SOAP vs. RESTful Web Services 8](#_Toc417050016)

[SOAP 8](#_Toc417050017)

[Anatomy of a SOAP Message 8](#_Toc417050018)

[Web Services Description Language (WSDL) 8](#_Toc417050019)

[Universal Description, Discovery, and Integration (UDDI) 8](#_Toc417050020)

[The History of the Web Service 8](#_Toc417050021)

[Web Services Life Cycle 9](#_Toc417050022)

[Servlets 9](#_Toc417050023)

[Why servlets? 9](#_Toc417050024)

[get and post and their doGet and doPost methods 9](#_Toc417050025)

[Terms to Look Up 9](#_Toc417050026)

[Code Examples 10](#_Toc417050027)

[JSF Guessnumber Program 10](#_Toc417050028)

[SOAP Request (HTTP) 10](#_Toc417050029)

[SOAP Response (HTTP) 10](#_Toc417050030)

[WSDL Example 11](#_Toc417050031)

[Images and Such 11](#_Toc417050032)

[Remaining things to lookup 11](#_Toc417050033)

# JavaServer Faces (JSF)

## What is JSF?

JSF, or Java Server Faces, is a server-side user interface component framework for Java-based web applications. Basically, it’s the web UI that can be developed for web-based Java applications. JSF is a **specification** and **reference implementation** for a web application development framework. It is made up of:

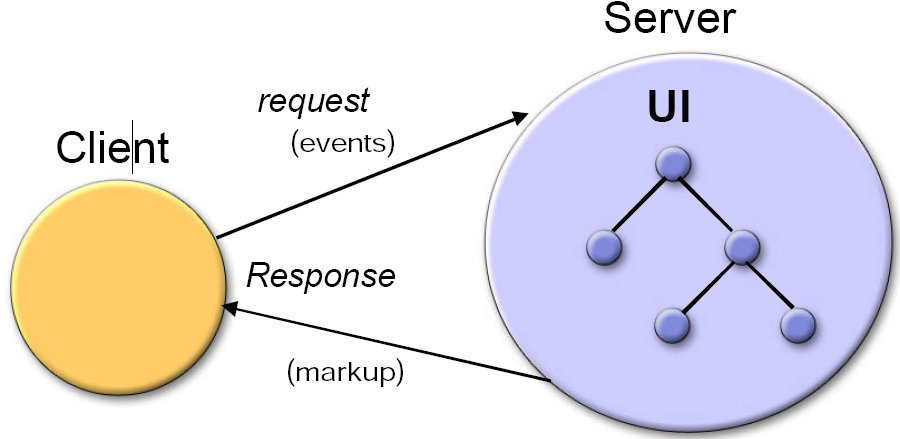
* Components
* Events
* Validators
* Back-end-Data Integration

JSF is designed to be leveraged by a variety of tools, such as Netbeans, RAD, Eclipse, etc. JSF makes use of an **events-based** interaction model, instead of the traditional request/response model, and provides a built-in **validation framework**, basic page **navigation support**, and many other features.

### JSF Pages

A JSF Page is represented by a tree of UI components, called a **view**. It’s easy to use,

### What JSF Looks Like



## Why JSF?

Why use JSF? Because it uses the **model-view-controller** architectural pattern, it’s easy to use, supports **client device independence**, also known as being **responsive** to the client’s device, regardless of if it’s a phone or tablet or TV, it has a huge vendor and industry support, and has a built-in UI component model. JSF also offers a finer-grained **separation of behavior and presentation** than other alternatives, treating UI elements as **stateful objects**. It even works with any presentation technology, and doesn’t restrict you to a particular scripting technology or markup language.

### Model-View Controller

## The JSF Life Cycle

The JSF “life cycle” begins when a client makes a request for a page. The requested **view** is then built by JSF, taking into consideration changes of state that were saved from the previous **postback**. Once **postback** is performed, the JSF implementation must perform **validation** and **conversion**.

### The Lifecycle Phases

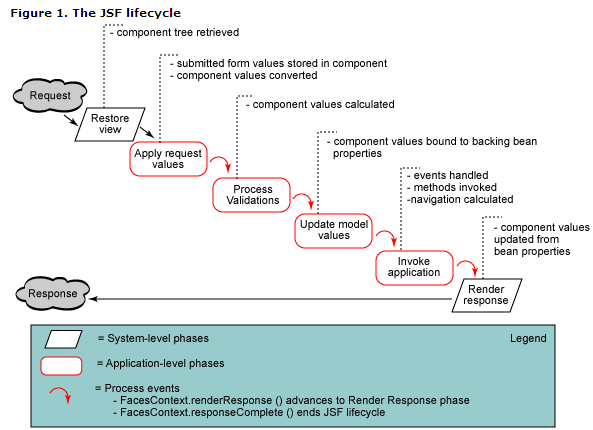
* **Restore View (Reconstitute Component Tree)**
  + JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request. During this phase, the JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance. The FacesContext instance will now contains all the information required to process a request.
* **Apply request values**
  + After the component tree is created/restored, each component in component tree uses decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.
  + If any decode methods / event listeners called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.
* **Process validations**
  + During this phase, the JSF processes all validators registered on component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component.
  + If the local value is invalid, the JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and display the same page again with the error message.
* **Update model values**
  + After the JSF checks that the data is valid, it walks over the component tree and set the corresponding server-side object properties to the components' local values. The JSF will update the bean properties corresponding to input component's value attribute.
  + If any updateModels methods called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.
* **Invoke application**
  + During this phase, the JSF handles any application-level events, such as submitting a form / linking to another page.
* **Render Response**
  + During this phase, the JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as the JSP container executes the page. If this is not an initial request, the component tree is already built so components need not to be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page.
  + After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.

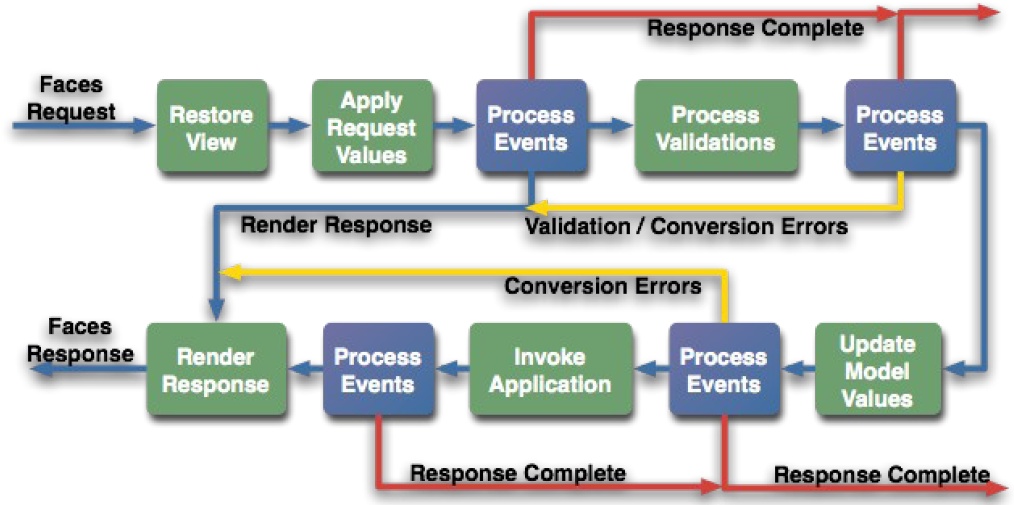
### JSF Request Processing

The JSF Life Cycle handles two types of requests, the **initial request** and **postback**.

* **Initial Request** – This occurs when a user requests the page for the first time. At this point, the lifecycle only executes the **restore view** and **render response** phases. It doesn’t execute the rest because there’s no need for them. The user hasn’t submitted any additional information to be processed.
* **Postback** – **Postback** occurs when a user submits a form contained on a page, which was previously loaded into the browser as a result of an **initial request**. At this point, the lifecycle executes all the phases.

### JSF Lifecycle





## How JSF Works

### Facelets

Facelets is the lightweight page declaration language that’s used to build JSF **views** using HTML-style **templates**, which are then used to build **component trees**. Facelets use **XHTML** to create web pages, and have their own “tag” libraries, in addition to JSF and JSTL tag libraries. It allows for templating for various components and pages.

#### Advantages

Facelets reduce the amount of time and effort that’s needed for development and deployment. Templating and composite components make it easy to re-use code, components are extensible, and it has a very fast compilation time, compile-time EL validation, and high-performance rendering.

#### Templating & Inclusion

JSF offers two features that streamline the creation of new pages; **templating** and **inclusion**. Templating is a way of establishing a common theme or layout throughout your pages, by re-using a **template,** often used in headers, footers, sidebars, or any other common elements that are re-used often in a site. There are four facelet tags that are used to build a page from a template:

1. ui:insert – This tag is used to declare a named content element that will be later defined by another facelet. In essence, it creates the structure, without content. The ui:define tag relies on this tag to tell it what content gets filled in.
2. ui:define – This templating tag defines the content that will be inserted into the template defined by ui:insert.
3. ui:include – This tag is used on the server-side to “**include**” files within a page as it is rendered. An example of this would be loading an additional html file within a page, for a page-within-a-page.
4. ui:composition – This tag defines content that will be included in another Facelet. Content that’s outside of this tag set will be ignored by the Facelets view handler.

### Event Handling

#### Page Navigation

##### Page Forwarding vs. Page Redirection

## Application Scopes

### How Scopes Work

As long as a user stays on a page, the values of components are remembered, even when the page redisplays. When the user leaves the page, component values then disappear. In order to make values available to other pages, or to make the values available to the same page if the user returns later, the values must be stored.

When you create a project from the IDE, the IDE creates three managed beans for storing values:

* RequestBean1
* SessionBean1
* ApplicationBean1

For more on Beans, see <pageno>.

### Application

Application scope lasts until the server stops the application. Values stored in an application bean are available to every session and every request that uses the same “application map”. It’s generally used for things that are used by all users, or track across multiple users (like a counter tracking users, or a drop-down list of measurement types that all use).

### Session

Session scope begins when a user first accesses a page in the web app, and ends when the session either times out from inactivity, or the web app invalidates the session (by calling session.invalidate()). This would see use in storing login info, or a shopping cart.

### Request

Request scope begins when the user submits the page, and ends when the response is fully rendered. This would be used for things that are only needed in a single request/response cycle, like a timestamp.

### View

## Beans

### What a Bean Is

A JavaBeans object is an object in Java that has all private properties (meaning getters and setters are used), a public, no-argument constructor, and implements “Serializable”. The last part means that the bean can be written to streams, things like files, databases, anything.

### Binding to Managed Beans

## SessionMap, ApplicationMap

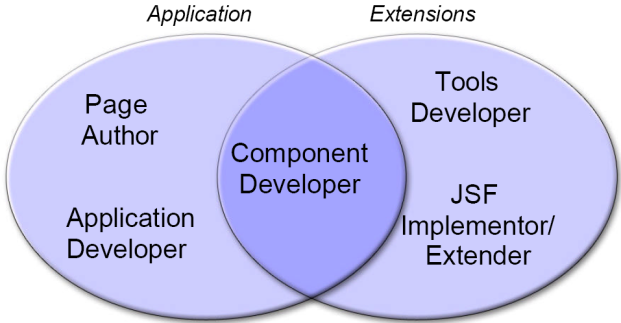
## Using JavaScript in JSF

## JSF 2.x AJAX support

## JavaMail

## GET parameters

## Developer Roles



### Page Author

A page author creates the user interface of the web application, and should generally be familiar with the markup languages that are being used. They assemble prebuilt components.

### Component Writer

The component writer creates reusable components, renderers, and libraries. **Components** are render-independent properties. **Renderers** are render-dependent properties.

# JDBC, Data Sources

## loading driver

## getting a connection

## preparing SQL statements

## running the query/update

## getting the results

## Using a data source to access and commit changes to a database

# XML

## Why XML?

## Difference between HTML and XML

## Well-formedness vs. Validity

## Elements vs. Attributes

## XML Schema

## XML vs. JSON

# Web Services

## What is a Web Service?

A web service is a program that can interact with another program, using (going through) standardized web protocols, such as HTTP. It allows for interoperable machine-to-machine interaction over a network. It uses an interface described in a **machine-processable** format (in this case, **WSDL**). The most commonly-known web service is the humble API.

## Why Use Web Services?

A web service allows for the sharing of information through the web without a terrible amount of work. A good example of it being used well is with Amazon, where users of their web services can allow their users to view content and buy products from Amazon without ever actually visiting Amazon.

Beyond all of that, web services allow for **integration and interoperability** at low cost, reducing the requirements for communication down to the “lowest common denominator”, being **XML-based protocols**. Web services are **relatively easy to use**, and are **reusable.** They are **ubiquitous**, able to be used regardless of where your system is.

## Problems & Concerns

* Reliability / Consistency
* Security
* Authentication
* Privacy
* Billing
* Reuse
* Performance
* Incompatible implementations of standards

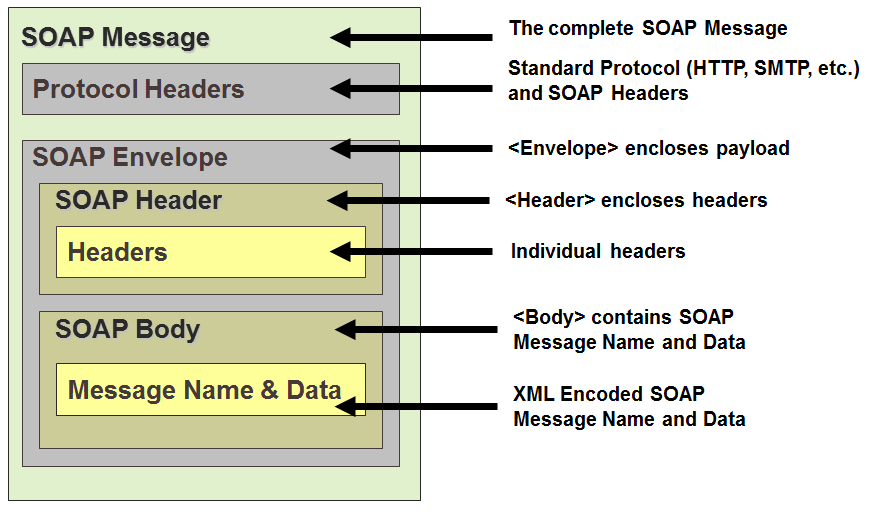
## SOAP vs. RESTful Web Services

### SOAP

SOAP stands for **Simple Object Access Protocol**, and is a **remote procedure call**-based (**RPC**) method. Systems can interact with a web service using SOAP-based messages, which is usually described in the service’s description. This description is usually conveyed using HTTP, with an XML serialization. SOAP uses a lightweight, XML-based messaging format, and builds on W3C XML standards, as well as the IETF HTTP standard. **SOAP can work with any operating system, programming language, and platform**.

<todo> Lim has a youtube link, check it out.

### Anatomy of a SOAP Message



## Web Services Description Language (WSDL)

Web Services Description Language, or WSDL, is a method by which web services describe **what they are**, **where they can be found**, and **how they should be used**. This is the primary interface of a web service.

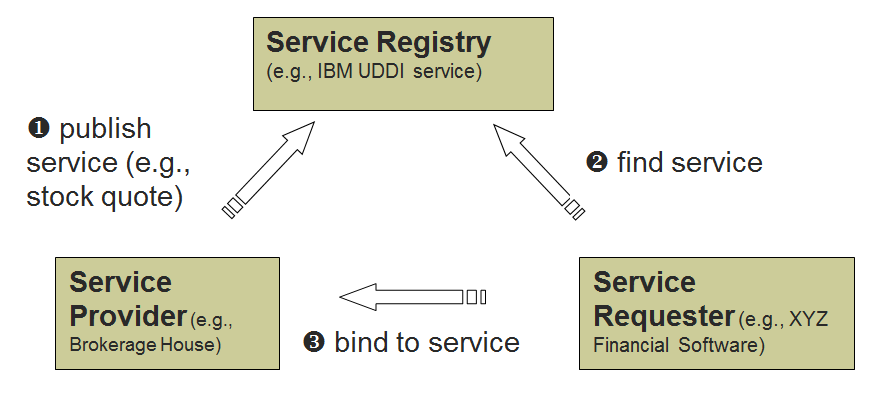
## Universal Description, Discovery, and Integration (UDDI)

Universal Description, Discovery, and Integration (UDDI) allows companies to find publically available web services on the internet, or on corporate intranets. Basically, think of them as yellow pages, making it easier to share business information across the web.

## The History of the Web Service

* 2002 (within the firewall)
  + Simplified app integration
  + Increased developer productivity
* 2004 (contained external users)
  + Simplified business-partner connectivity
  + Richer app functionality
  + Subscription-based services
* 2006 to 2008 (fully dynamic search and use)
  + Casual / ad-hoc use of services
  + New business models possible
  + Commoditization of software
  + Pervasive use in nontraditional devices

## Web Services Life Cycle



# Servlets

## Why servlets?

## get and post and their doGet and doPost methods

# Terms to Look Up

* Application Map
* Component Trees
* How SOAP’s description is found
* Extensible Component and Rendering architecture

Web Services, Security, AJAX, and other materials covered before test

# Code Examples

## Templating

### Facelets Template

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:h="http://java.sun.com/jsf/html"

      xmlns:ui="http://java.sun.com/jsf/facelets">

<h:head>

<title><ui:insert name="title" /></title>

</h:head>

<body>

<h2><ui:insert name="header" /></h2>

<ui:insert name="message" />

</body>

</html>

### Facelets Client (uses Template)

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:ui="http://java.sun.com/jsf/facelets">

<h:head />

<body>

<ui:composition template="template.xhtml">

  <ui:define name="title">Welcome</ui:define>

  <ui:define name="header">Hello World</ui:define>

  <ui:define name="message">How are you today?</ui:define>

</ui:composition>

</body>

</html>

### HTML Output

<html>

  <head>

    <title>Welcome</title>

  </head>

  <body>

    <h2>Hello World!</h2>

    How are you today?

  </body>

</html>

## JSF Guessnumber Program

## SOAP Request (HTTP)

POST /StockQuote HTTP/1.1   
Host: www.stockquoteserver.com   
Content-Type: text/xml   
Content-Length: 323   
SOAPAction: “www.stockquoteserver.com/GetLastTradePrice”

<?xml version=“1.0” encoding=“utf-8”?>

<SOAP-ENV:Envelope   
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"   
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>  
 <m:GetLastTradePrice xmlns:m="Some-Namespace-URI">  
 <symbol>DIS</symbol>  
 </m:GetLastTradePrice>  
 </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>

## SOAP Response (HTTP)

HTTP/1.1 200 OK  
Content-Type: text/xml; charset=utf-8

Content-Length: nnnn

<?xml version=“1.0” encoding=“utf-8”?>

<SOAP-ENV:Envelope  
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

<SOAP-ENV:Body>  
 <m:GetLastTradePriceResponse   
 xmlns:m="Some-Namespace-URI">  
 <Price>24.5</Price>  
 </m:GetLastTradePriceResponse>  
 </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>

## WSDL Example

<?xml version="1.0"?>  
<serviceDescription>  
 <soap >  
 <service>  
 <addresses>  
 <address uri="http://localhost//HelloWorld.asmx"/>  
 </addresses>  
 <requestResponse name="HelloWorld" soapAction="http://tempuri.org/HelloWorld">  
 <request ref="s0:HelloWorld"/>  
 <response ref="s0:HelloWorldResult"/>  
 </requestResponse>  
 </service>  
 </soap>  
</serviceDescription>

# Images and Such

# Remaining things to lookup

Patterns

MVC

DAO

CDI

horizontal rule

Exam Questions/Format

Miniature versions of the questions given in Lab3, Assignment 3, and Lab 4.

Study the following code and answer the questions given below. ... What would the code produce when compiled and run? Explain.

...

Study the following code and answer the questions given below. ... In general, what does the program do? Don’t give line-by-line analysis, give the overall effect.

Compare and contrast\_\_\_\_\_\_\_\_\_\_\_ with respect to \_\_\_\_\_\_\_\_\_\_\_\_\_ (e.g., get and post requests)

Given a form in an XHTML page, write a JSF app that processes the form and return the result.

Write a JDBC segment to retrieve all employees who make more than $30000.

Using a session object, validate a user's access to \_\_\_\_\_\_

         Big pictures questions

- AJAX in JSF 2.x (read and code simple AJAX using f:ajax tag)

- Web Services materials covered in Lab4

- SQL Injection and Cross-site Scripting