

# Java Programming

# Why JSF?

- JavaServer Faces (JSF) is a new standard Java framework for building Web applications.
- **Java** specification for building component-based **user interfaces for web applications**
- JSF offers a clean separation between behavior and presentation for web applications.

# What is JSF?

- Java Server Faces (JSF) technology is a **front end** framework which makes the creation of user interface components easier by reusing the UI components.
- JSF is designed based on the Model View Controller pattern (MVC) which segregates the presentation, controller and the business logic.
- **UI Components:** Text fields, list boxes, checkboxes, labels, panels, radio buttons, and other elements

# JSF Features

- Component Based Framework.
- JSP is based on the Model-View-Controller concept
- Ease and Rapid web Development.
- Default Exception Handling.
- JSF separates the functionality of a component from the display of the component.

# Why not JSF?

- JSF forces you to mix Java and xhtml code for the same feature.
- JSF project can be easily become too complex to maintain

# JSF program

- JSF provides a standard HTML tag library which are rendered into corresponding html output.
- In order to use these these tags we need to use the following namespaces of URI in html node.
- `<html  
xmlns="http://www.w3.org/1999/xhtml"  
xmlns:h="http://java.sun.com/jsf/html" >`

JSF tags	HTML tags
h:inputText	<input type="text">
h:outputText	Plain text
h:form	<form>
h:commandButton	<input type=button> value can be "submit", "reset", or "image"
h:inputSecret	<input type="password">
h:inputTextarea	<textarea>
h:inputHidden	<input type="hidden">
h:dataTable	<table>
h:outputLabel	<label>
h:panelGrid	<table> element with <tr> and <td> elements
h:selectOneRadio	<input type="radio">
h:selectBooleanCheckbox	<input type="checkbox">

JSF tags are  
similar to  
html tags  
with minor  
variations



# JSF-Example code 1

## java class and X-html

@ManagedBean(name="hello")

This is the name for the object of HelloWorld class

Later can use in JSF tag to fetch its variables as #{hello.s1}

```
@ManagedBean(name="hello")
public class HelloWorld
{
    private String s1 = "Hello World!!";
}
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<html
xmlns="https://www.w3.org/1999/xhtml"
xmlns:h="https://java.sun.com/jsf/html">
  <h:head>
    <title>Hello World JSF Example</title>
  </h:head>
  <h:body>
    #{hello.s1}
    <br />
    <br />
  </h:body>
</html>
```

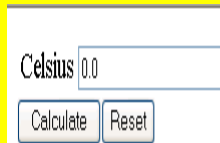
Use <h:head> instead of <head>

Access the java class HelloWorld  
variable s1 via java object "hello"  
declared using @ManagedBean



**@ManagedBean(name="temperatureConvertor")**

```
public class TemperatureConvertor {
    private double celsius;
    private double fahrenheit;
    private boolean initial= true;
    public double getCelsius() {
        return celsius; }
    public void setCelsius(double celsius) {
        this.celsius = celsius; }
    public double getFahrenheit() {
        return fahrenheit; }
    public boolean getInitial(){
        return initial; }
    public String reset (){
        initial = true;
        fahrenheit =0;
        celsius = 0;
        return "reset";
    }
    public String celsiusToFahrenheit(){ initial
= false;
    fahrenheit = (celsius *9 / 5) +32;
    return "calculated"; } }
```



```
<html>
<title>Celsius to Fahrenheit Convertor</title>
<h:body>
<h:form>
<h:outputLabel value="Celsius">
</h:outputLabel>
<h:inputText
value="#{temperatureConvertor.celsius}">
</h:inputText>
<h:commandButton action=
"#{temperatureConvertor.celsiusToFahrenheit}"
value="Calculate">
</h:commandButton>
<h:commandButton action=
"#{temperatureConvertor.reset}" value="Reset">
</h:commandButton>
</h:form>
<h3> Result </h3>
<h:outputLabel value="Fahrenheit ">
</h:outputLabel>
<h:outputLabel
value="#{temperatureConvertor.fahrenheit}">
</h:outputLabel> </h:body> </html>
```

## Example 2: jdbc connectivity

```
// index.xhtml
html xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head> <title>User Form</title> </h:head>
<h:body>
<h:form>
<h:outputLabel for="username" value="User Name "/>
<h:inputText id="username" value="#{user.userName}"> </h:inputText><br/>
<h:outputLabel for="email" value="Email ID "/>
<h:inputText id="email" value="#{user.email}">
</h:inputText><br/><br/>
<h:commandButton action="#{user.submit()}" value="submit"/>
</h:form>
</h:body>
</html>
```

## Example 2: jdbc connectivity

```
// User.java
@ManagedBean(name="user")
public class User {
    String userName;
    String email;
    public String getUser_name() {
        return userName; }
    public void setUser_name(String user_name)
    {
        this.userName = user_name; }
    public String getEmail() {
        return email; }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

```
public boolean save(){
    int result = 0;
    try{
        String url="jdbc:oracle:thin:@localhost:1521:XE";
        String user ="SYSTEM";
        String pwd ="oracle";
        Connection
        con=DriverManager.getConnection(url,user,pwd);
        PreparedStatement stmt = con.prepareStatement(
            "insert into user(name,email) values(?,?)");
        stmt.setString(1, this.getUserName());
        stmt.setString(2, this.getEmail());
        result = stmt.executeUpdate();
    }catch(Exception e){ System.out.println(e); }
    if(result == 1){ return true; }
    else return false;
}

public String submit(){
    if(this.save()){ return "response.xhtml";
    }else return "index.xhtml";
    }
}
```

## Example 2: jdbc connectivity

```
// response.xhtml
```

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

```
xmlns:h="http://xmlns.jcp.org/jsf/html">
```

```
<h:head>
```

```
<title>Response Page</title>
```

```
</h:head>
```

```
<h:body>
```

```
<h1><h:outputText value="Hello #{user.userName}"/></h1>
```

```
<h:outputText value="Your Record has been Saved Successfully!"/>
```

```
</h:body>
```

```
</html>
```

# Web services

- A web service makes software application resources available over networks using standard technologies.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems
- Web services are based on standard interfaces and hence they can communicate even if they are running on different operating systems and are written in different languages.
- Therefore, Web services are an excellent approach for building distributed applications that must incorporate diverse systems over a network.

# What are Web services?

- **Web services** are client and server applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP).
- web services provide a standard means of interoperating between software applications running on a variety of platforms and frameworks.
- Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions, XML.
- Web services can be combined in a loosely coupled way to achieve complex operations.
- Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.



- A web service:
  - Publicly describes its own functionality through a **WSDL** file
    - WSDL is an XML, and it stands for Web Service Description Language. WSDL describes all the methods available in the web service, along with the request and response types. It describes the contract between service and client.
  - Communicates with other applications via XML messages, often formatted with **SOAP**
  - Employs a standard network protocol such as **HTTP**

# Types of web services

- There are two types of web services:
  - SOAP Web Services
  - REST Web Services



# SOAP

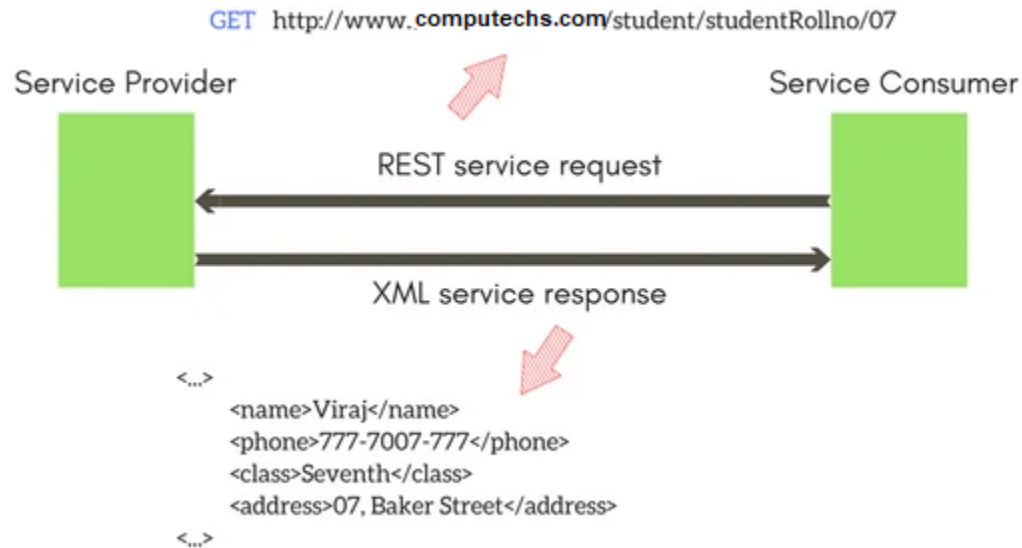
- SOAP is an XML-based protocol.
- SOAP stands for **Simple Object Access Protocol**.
- SOAP was intended to be a way to do remote procedure calls to remote objects by sending XML over HTTP.



# REST Web Services

- The **REST** stands for **Representational State Transfer**.
- REST is not a set of standards or rules, rather it is a style of software architecture.
- The applications which follow this architecture are referred to as **RESTful**
- REST locates the resources by using URL and it depends on the type of **transport** protocol(with HTTP - GET, POST, PUT, DELETE,...) for the actions to be performed on the resources.
- while requesting the data from a website, the data should be in a browser readable format, which is **HTML**, while in case of the REST API, response can be anything like **XML/JSON** or any other media type.

# REST Web Services



# Difference between REST and SOAP

## REST

- REST is a style of software architecture.
- REST can use SOAP because it is a concept and can use any protocol like HTTP, SOAP etc.
- REST uses URI to expose business logic.
- REST inherits security measures from the underlying transport protocols.
- REST accepts different data formats like, Plain Text, HTML, JSON, XML etc.

## SOAP

- SOAP is a protocol or a set of standards.
- SOAP cannot use REST because it itself is a protocol.
- SOAP uses the service interface to expose business logic.
- SOAP defines its own security layer.
- SOAP only works with XML format.