

# CSC584 Enterprise Programming

---

CHAPTER 6 – ENTERPRISE JAVABEAN (EJB)

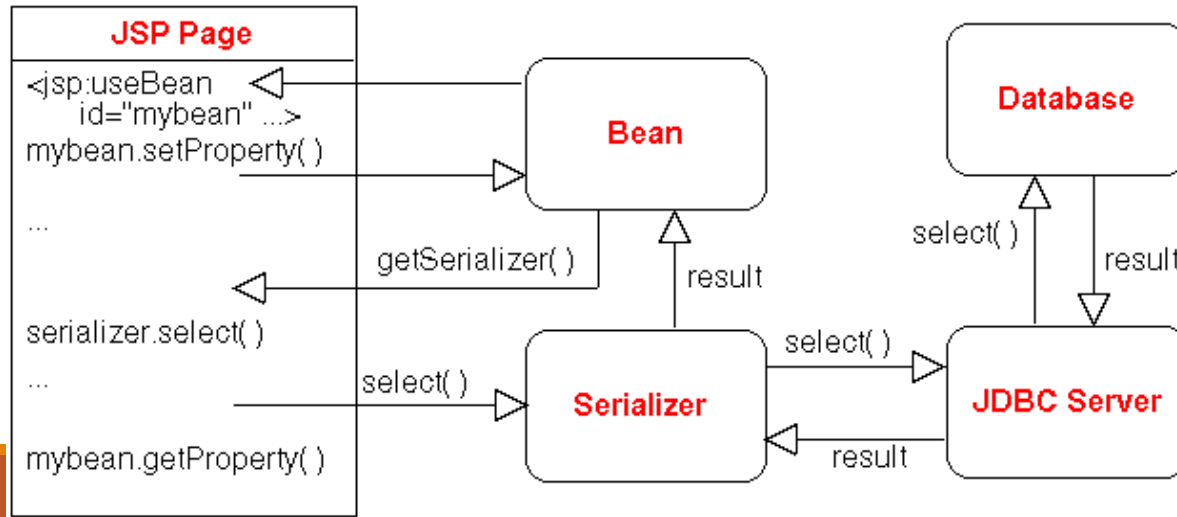
# Chapter 6 Outline

## **Enterprise JavaBean (EJB) components**

- Define Java Bean -create java bean
- Describe the custom bean properties and events
- Describe types of EJB - session beans, entity beans and message driven beans

# Introduction to JavaBeans

- What are JavaBeans?
  - Software components written in Java
  - Connect and Configure Components
  - Builder Tools allow connection and configuration of Beans
  - Begins 'Age of Component Developer'
  - Bringing Engineering methods to Software Engineering (e.g. electronics...)



# Definitions Java Bean

---

- A reusable software component that can be manipulated visually in a 'builder tool'. (from JavaBean Specification)
- The JavaBeans API provides a framework for defining reusable, embeddable, modular software components.
- Purpose: Store Data
- Simple Object, requires no argument constructor
- Properties accessible via get & set methods

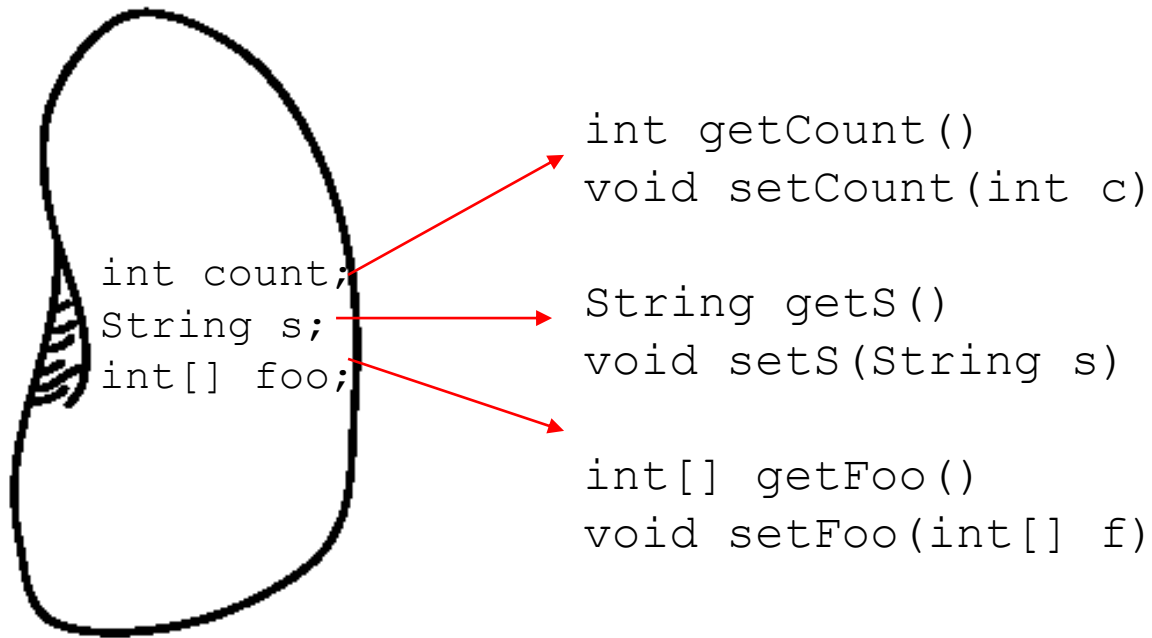
# Java Bean - example

---

- For a "foo" property, a java bean will respond to
  - `type getFoo()`
  - `void setFoo(type foo)`
- For a boolean "bar" property, a java bean will respond to
  - `boolean isBar()`
  - `void setBar(boolean bar)`

# Java Bean

---



```
// MagicBean.java
/* A simple bean that contains a single
 * "magic" string.
 */
public class MagicBean {
    private String magic;
    public MagicBean() { // default constructor
        magic = "Woo Hoo";
    }
    public MagicBean(String magic) { // normal constructor
        this.magic = magic;
    }
    public String getMagic() {
        return(magic);
    }
    public void setMagic(String magic) {
        this.magic = magic;
    }
}
```

# Java Beans

```
<jsp:useBean id="myBean" class="com.foo.MyBean"  
scope="request"/>  
<jsp:getProperty name="myBean" property="lastChanged" />  
<jsp:setProperty name="myBean" property="lastChanged"  
value="<%= new Date()%>"/>
```

## Example

- `<jsp:useBean id="bean" class="MagicBean" />`
- `<jsp:getProperty name="bean" property="magic" />`
- `<jsp:setProperty name="bean" property="magic"  
value="Hoo"/>`



```
<!-- bean.jsp -->
<hr>
<h3>Bean JSP</h3>
```

```
<p>Have all sorts of elaborate, tasteful HTML ("presentation") surrounding the data we pull
off the bean. </p>
```

Behold -- I bring forth the magic property from the Magic Bean...

```
<!-- bring in the bean under the name "bean" -->
```

```
<jsp:useBean id="bean" class="MagicBean" />
```

```
<table border=1>
```

```
  <tr>
```

```
    <td bgcolor=green><font size=+2>Woo</font> Hoo</td>
```

```
    <td bgcolor=pink>
```

```
      <font size=+3>
```

```
        <!-- the following effectively does bean.getMagic() -->
```

```
        <jsp:getProperty name="bean" property="magic" />
```

```
      </font>
```

```
    </td>
```

```
    <td bgcolor=yellow>Woo <font size=+2>Hoo</font></td>
```

```
  </tr>
```

```
</table>
```

```
<!-- pull in content from another page at request time with a relative URL ref to another
page -->
```

```
<jsp:include page="trailer.html" flush="true" />
```

```
public class HelloBean extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<html>");
        out.println("<head>");
        String title = "Hello Bean";
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=white>");
        out.println("<h1>" + title + "</h1>");
        out.println("<p>Let's see what Mr. JSP has to contribute...</p>");
        request.setAttribute("foo", "Binky");
        MagicBean bean = new MagicBean("Peanut butter sandwiches!");
        request.setAttribute("bean", bean);
        RequestDispatcher rd = getServletContext().getRequestDispatcher("/bean.jsp");
        rd.include(request, response);
        rd.include(request, response);
        out.println("<hr>");
        out.println("</body>");
        out.println("</html>");
    }
    // Override doPost() -- just have it call doGet()
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response);
    }
}
```

## Using JavaBeans in JSP

---

To create an instance for a JavaBeans component, use the following syntax:

```
<jsp:useBean id="objectName"  
scope="scopeAttribute" class="ClassName" />
```

This syntax is equivalent to

```
<% ClassName objectName = new ClassName() %>
```

except that the scope attribute specifies the scope of the object.

# Scope Attributes

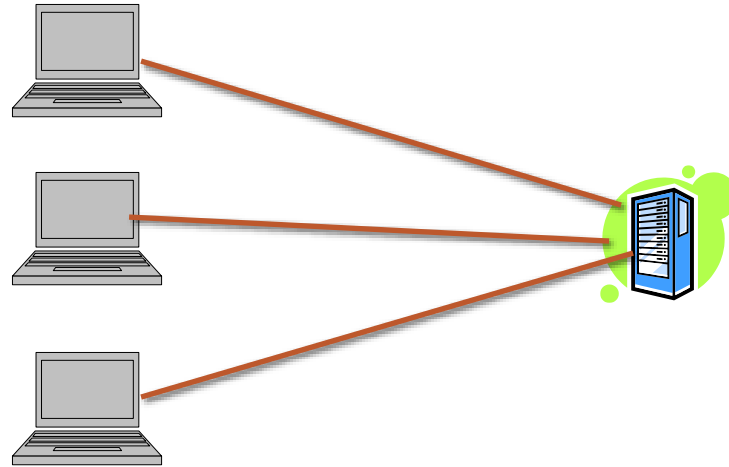
## **application**

session

page

request

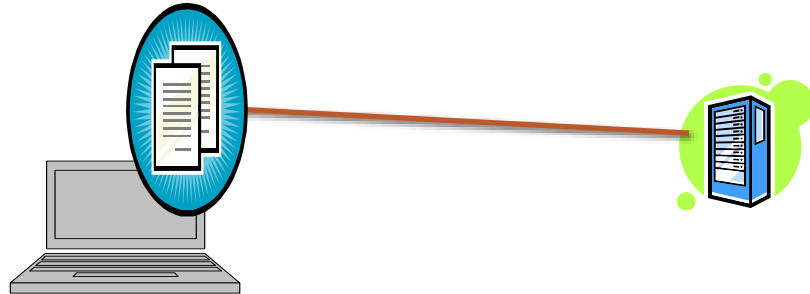
Specifies that the object is bound to the application. The object can be shared by all sessions of the application.



# Scope Attributes

application  
**session**  
page  
request

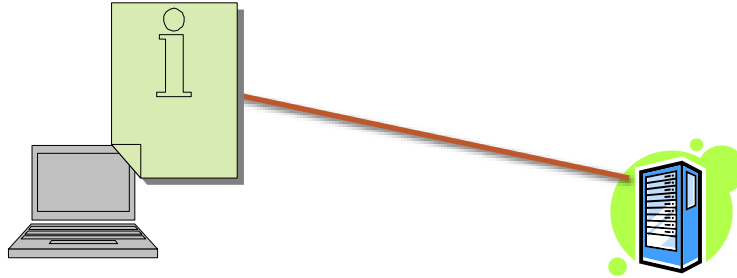
Specifies that the object is bound to the client's session. Recall that a client's session is automatically created between a Web browser and Web server. When a client from the same browser accesses two servlets or two JSP pages on the same server, the session is the same.



# Scope Attributes

application  
session  
**page**  
request

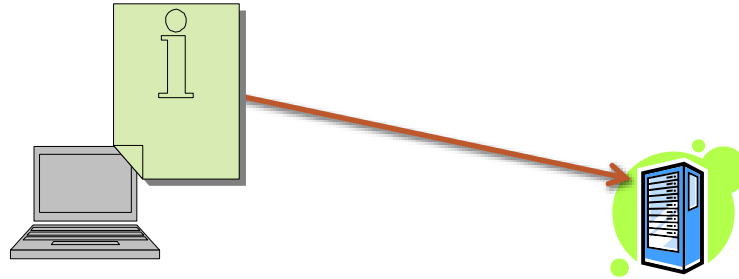
The default scope, which specifies that the object is bound to the page.



# Scope Attributes

application  
session  
page  
**request**

Specifies that the object is bound to the client's request.



# How Does JSP Find an Object

---

When `<jsp:useBean id="objectName" scope="scopeAttribute" class="ClassName"/>` is processed, the JSP engine first searches for the object of the class with the same id and scope. If found, the pre-existing bean is used; otherwise, a new bean is created.



# Another Syntax for Creating a Bean

---

Here is another syntax for creating a bean using the following statement:

```
<jsp:useBean id="objectName"  
  scope="scopeAttribute" class="ClassName" >  
  some statements  
</jsp:useBean>
```

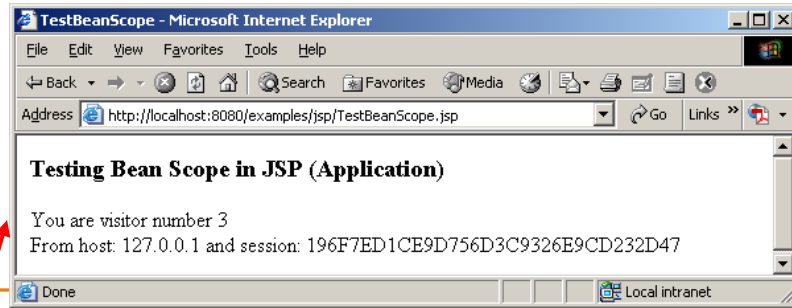
The statements are executed when the bean is created. If the bean with the same id and className already exists, the statements are not executed.

# Example: Testing Bean Scope

---

This example creates a JavaBeans component named Count and uses it to count the number of visits to a page.





```
<!-- TestBeanScope.jsp -->
<%@ page import = "model.Count" %>
<jsp:useBean id="count" scope="application"
class="model.Count">
</jsp:useBean>
<html>
<head>
    <title>TestBeanScope</title>
</head>
<body>
    <H3>Testing Bean Scope in JSP (Application)</H3>
    <% count.increaseCount(); %>
    You are visitor number <%= count.getCount() %>
<br>
    From host: <%= request.getRemoteHost() %>
    and session: <%= session.getId() %>
</body>
</html>
```

```
package model;
```

```
public class Count {
    private int count = 0;

    /** Return count property */
    public int getCount() {
        return count;
    }

    /** Increase count */
    public void increaseCount() {
        count++;
    }
}
```

# Getting and Setting Properties

---

By convention, a JavaBeans component provides the get and set methods for reading and modifying its private properties. You can get the property in JSP using the following syntax:

```
<jsp:getProperty name="beanId" property="age" />
```

This is equivalent to

```
<%= beanId.getAge() %>
```

# Getting and Setting Properties (cont.)

---

You can set the property in JSP using the following syntax:

```
<jsp:setProperty name="beanId" property="age" value="30" />
```

This is equivalent to

```
<% beanId.setAge(30); %>
```

# Associating Properties with Input Parameters

---

Often properties are associated with input parameters. Suppose you want to get the value of the input parameter named score and set it to the JavaBeans property named score. You may write the following code:

```
<% double score = Double.parseDouble(  
    request.getParameter("score")); %>  
  
<jsp:setProperty name="beanId" property="score"  
    value="<%= score %>" />
```

# Associating Properties with Input Parameters (cont.)

---

This is cumbersome. JSP provides a convenient syntax that can be used to simplify it as follows:

```
<jsp:setProperty name="beanId" property="score"  
    param="score" />
```

Instead of using the value attribute, you use the **param** attribute to name an input parameter. The value of this parameter is set to the property.

# Associating All Properties

---

Often the bean property and the parameter have the same name. You can use the following convenient statement to associate all the bean properties in beanId with the parameters that match the property names.

```
<jsp:setProperty name="beanId" property="*" />
```

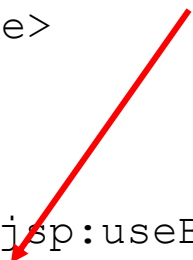


# Example: Computing Loan Using JavaBeans

Use JavaBeans to simplify Example 4.3 by associating the bean properties with the input parameters.

```
<!-- ComputeLoan.jsp -->
<html>
<head>
  <title>Compute Loan Using the Loan Class</title>
</head>
<body>
  <%@ page import = "model.Loan" %>
  <jsp:useBean id="loan" class="model.Loan"></jsp:useBean>
  <jsp:setProperty name="loan" property="*" />
  Loan Amount: <%= loan.getLoanAmount() %><br>
  Annual Interest Rate: <%= loan.getAnnualInterestRate() %><br>
  Number of Years: <%= loan.getNumberOfYears() %><br>
  <b>Monthly Payment: <%= loan.monthlyPayment() %><br>
  Total Payment: <%= loan.totalPayment() %><br></b>
</body>
</html>
```

Associating the bean properties with the input parameters.



# Enterprise Java Beans (EJB)

---

## Types of EJB s are:

- **Entity Beans**
  - They model: products, customers, data.
  - They have a persistent state.
  - They have been replaced with the Java Persistence API entities in EE 6 (they exist in previous versions, prior to EE5).
- **Session Beans**
  - They model: processes, coordinate the activities of EJBs
  - Do not have a persistent state.
- **Message Driven Beans** ( these beans require usage of Java Message Service API - JMS)

# Enterprise Java Beans

---

- Enterprise JavaBeans technology supports both **transient** and **persistent** objects.
- A **transient** object is called a *session bean*, and a **persistent** object is called an *entity bean*.

# Enterprise Java Beans

---

- Types of Entity beans:

```
graph LR; A[Entity Beans] --> B[Entity Beans that implement container managed persistence]; A --> C[Entity Beans that implement bean managed persistence];
```

Entity Beans

Entity Beans that  
implement container  
managed persistence

These beans are  
defined Independent of  
the database that they  
use

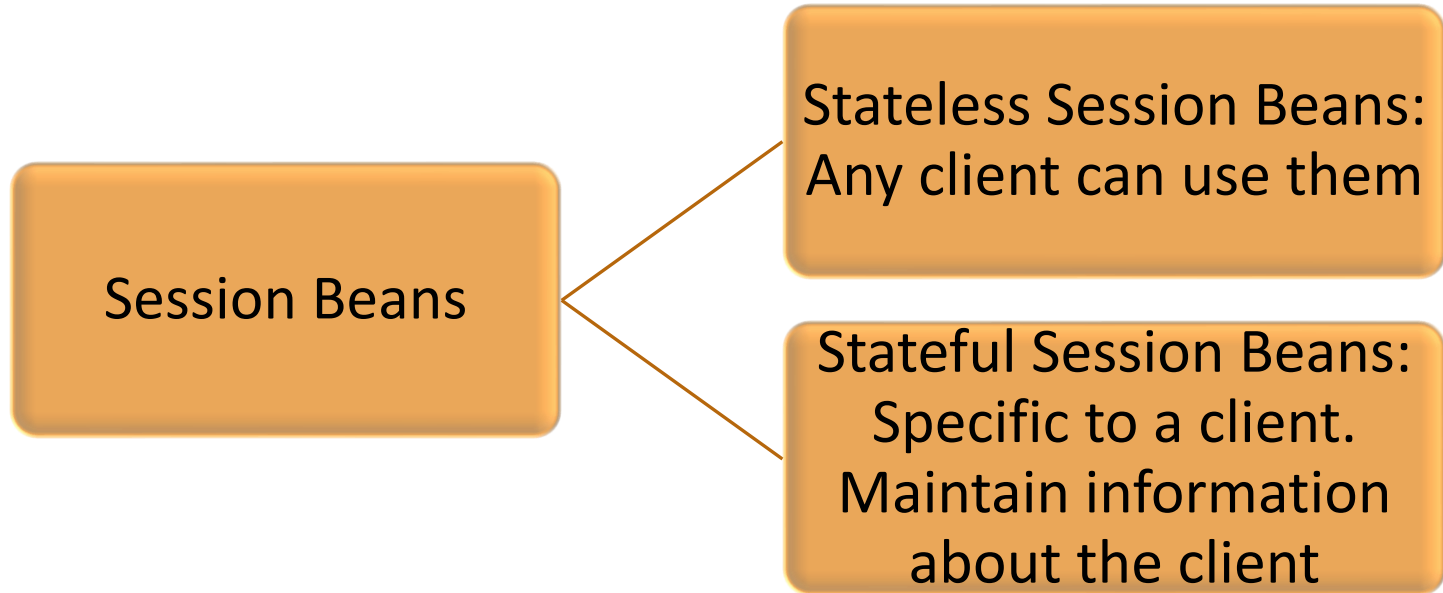
Entity Beans that  
implement bean  
managed persistence

These beans use  
explicit SQL  
statements relating to  
the database they use

# Enterprise Java Beans

---

- Types of session beans:



# Session Beans vs. Entity Beans

## Session Bean

- represent a business process, e.g. Billing credit card, trading stocks.
- is associated with one client and the life of the session bean is the life of the customer.
- do not survive from the server crashes

## Entity Bean

- representation of persistent data
- can be shared by multiple clients
- can read from DB and save back to DB
- has much longer life and can survive from server crashes

# Activation vs. Passivation

## Activation

- When a client needs to use a bean that has been passivated, an activation process occurred.
- The state of the bean is swapped in from the persistent storage

## Passivation

- If too many beans are instantiated, EJB container can passivate some of them
- the state of the bean is saved in a persistent store

# Stateless vs. Stateful

## Stateless

- no internal state
- can be pooled to service multiple client
- need not to handle activation and passivation
- examples: calculator

## Stateful

- possess internal state
- need to hand activation and passivation
- examples: shopping cart



# Container Managed Persistent vs. Bean Managed Persistent

## Container Managed

- EJB container is responsible for saving and retrieving bean's state
- Independent of data source
- Easy to develop

## Bean Managed

- Entity bean is responsible for saving bean's state.
- Less adaptive than container managed entity bean
- persistence need to be hard coded into the bean

# Example on JavaBean JSP

## 1. Client File Name: Login.jsp

```
<html>
  <head>
<title>JavaBean JSP</title>
  </head>
  <body>
    <h2>Using Java Beans with JSP</h2>
    <form method="get" action="Receive.jsp">
      Enter Username <input type="text" name="user"><br><br>
      Enter Password <input type="password" name="pass"><br>
      <input type="submit" name="Submit">
    </form>
  </body>
</html>
```

The HTML read user name and password from client and sends them to Receive.jsp

✕

— □ ✕

← → ↻ 🏠 ⓘ localhost:8080/Topic6/Login.jsp ⋮

# Using Java Beans with JSP

Enter Username

Enter Password

Submit

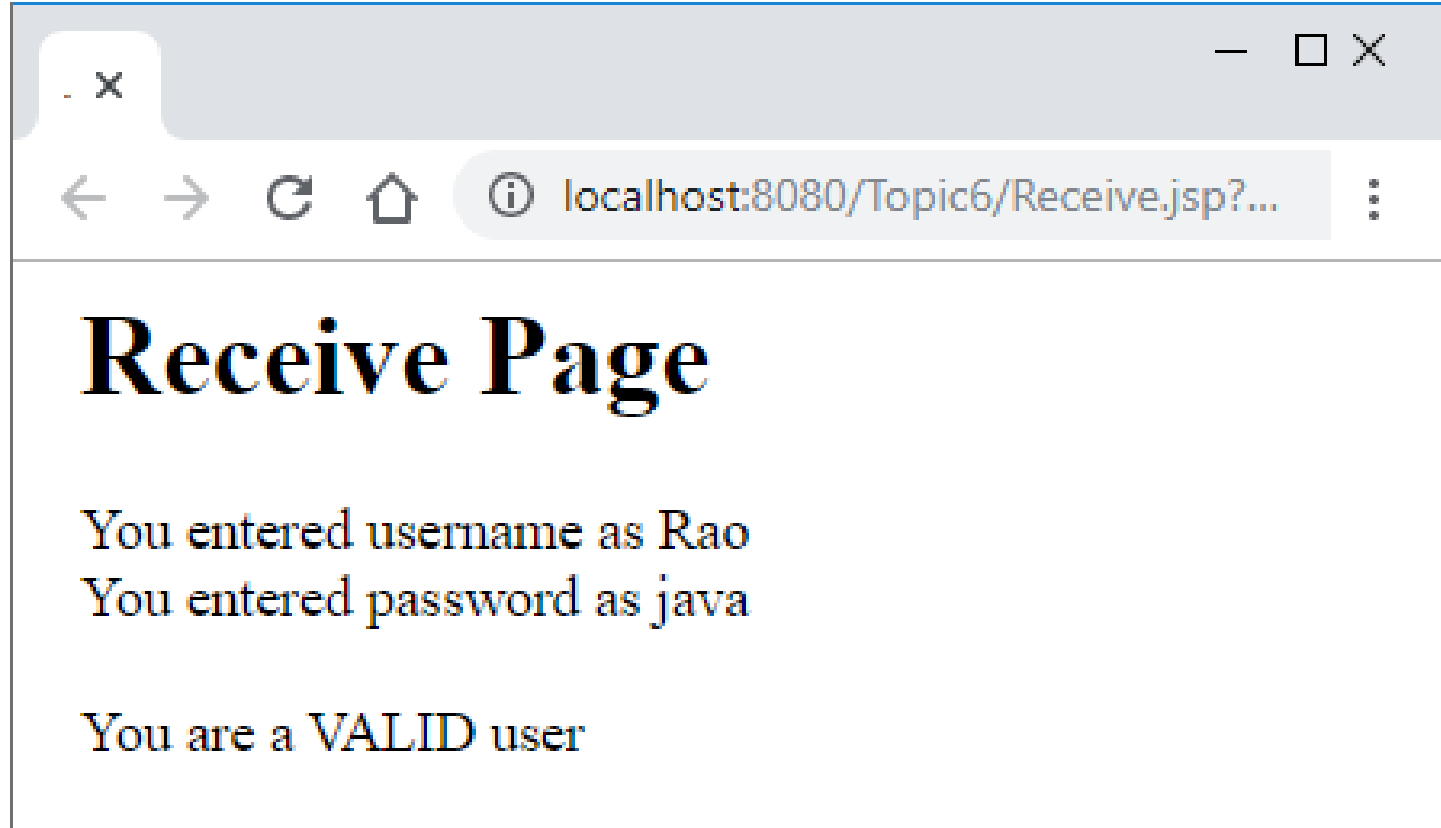
## 2. Server-side File Name: Receive.jsp

```
<html>
  <head>
    <title>Receive Page</title>
  </head>
  <body>
    <h1>Receive Page</h1>
    <jsp:useBean id="snr" class="model.ValidateBean" />
    <jsp:setProperty name="snr" property="user"/>
    <jsp:setProperty name="snr" property="pass"/>

    <% //<jsp:setProperty name="snr" property="*" /> this for all attributes%>

    You entered username as <jsp:getProperty name="snr" property="user"/><br>
    You entered password as <jsp:getProperty name="snr" property="pass"/><br>

    <br>
    You are a <%= snr.validate("Rao", "java")%> user<br>
  </body>
</html>1
```



### 3. Server-side File Name: ValidateBean.java

```
package model;
public class ValidateBean {
    String user;
    String pass;

    public ValidateBean() {}

    public String getUser() { return user;}
    public String getPass() {return pass;}

    public void setUser(String user) {
        this.user = user;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }

    public String validate(String s1, String s2){
        if(s1.equals(user) && s2.equals(pass))
            return "VALID";
        else
            return "INVALID";
    }
}
```

# Lab exercise

---

- Creating a Simple Web Application Using a MySQL Database.
  - Demonstrates how to create a simple web application that connects to a MySQL database server.
  - <https://netbeans.org/kb/docs/web/mysql-webapp.html>