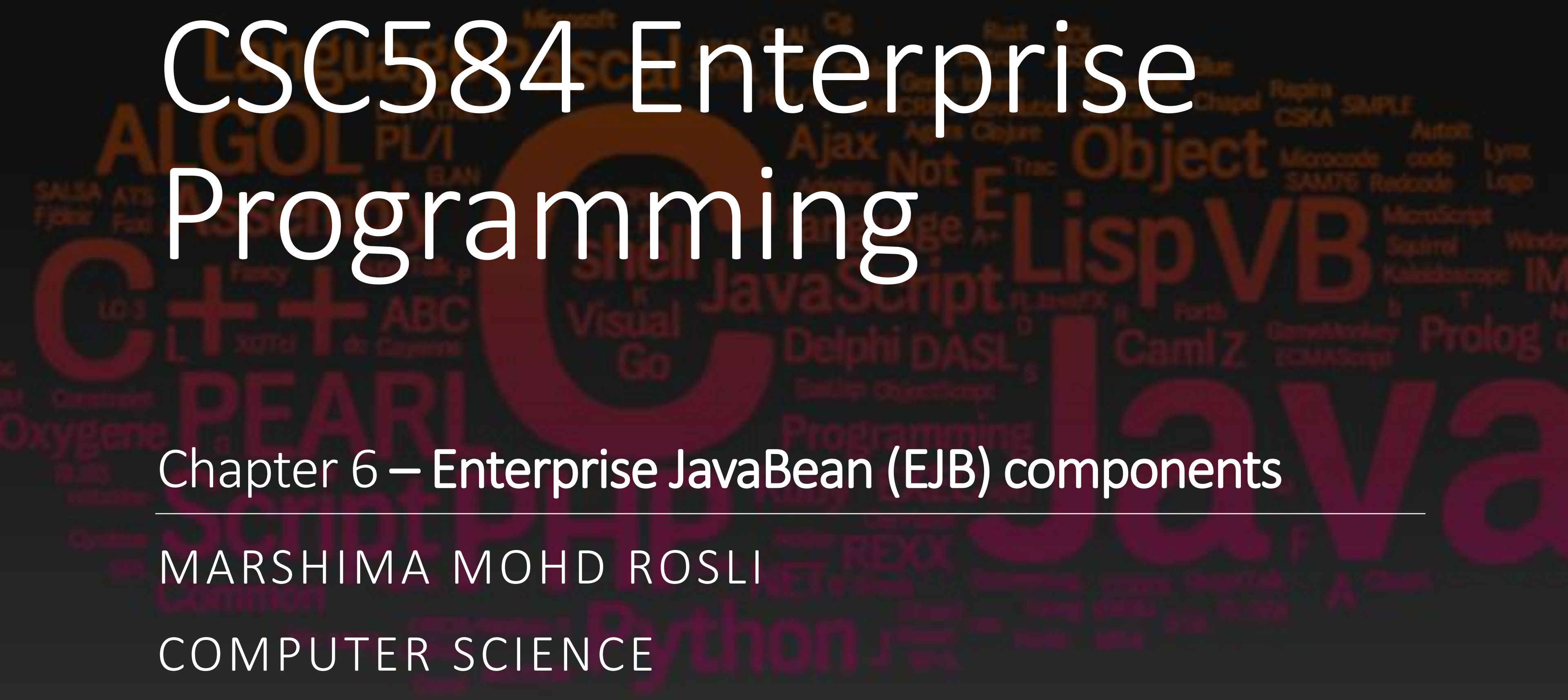


CSC584 Enterprise Programming

Chapter 6 – Enterprise JavaBean (EJB) components

MARSHIMA MOHD ROSLI

COMPUTER SCIENCE

Chapter 6 Outline

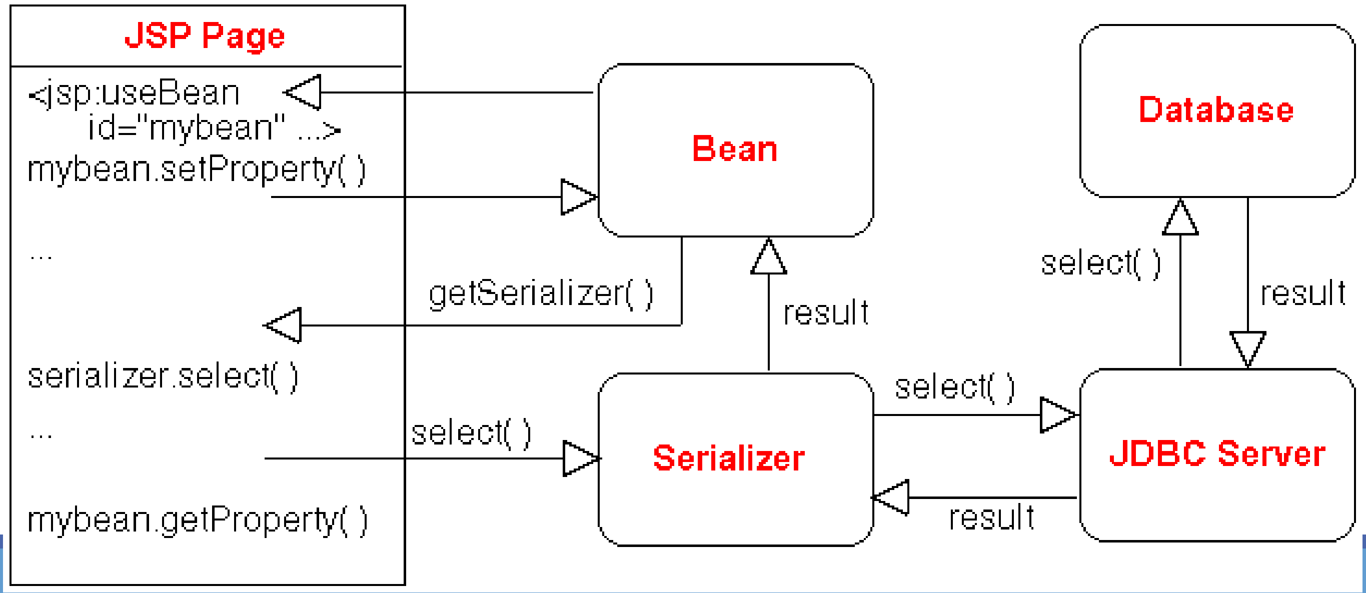
Enterprise JavaBean (EJB) components

- ❑ 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 Beans

- ❑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 Beans - example

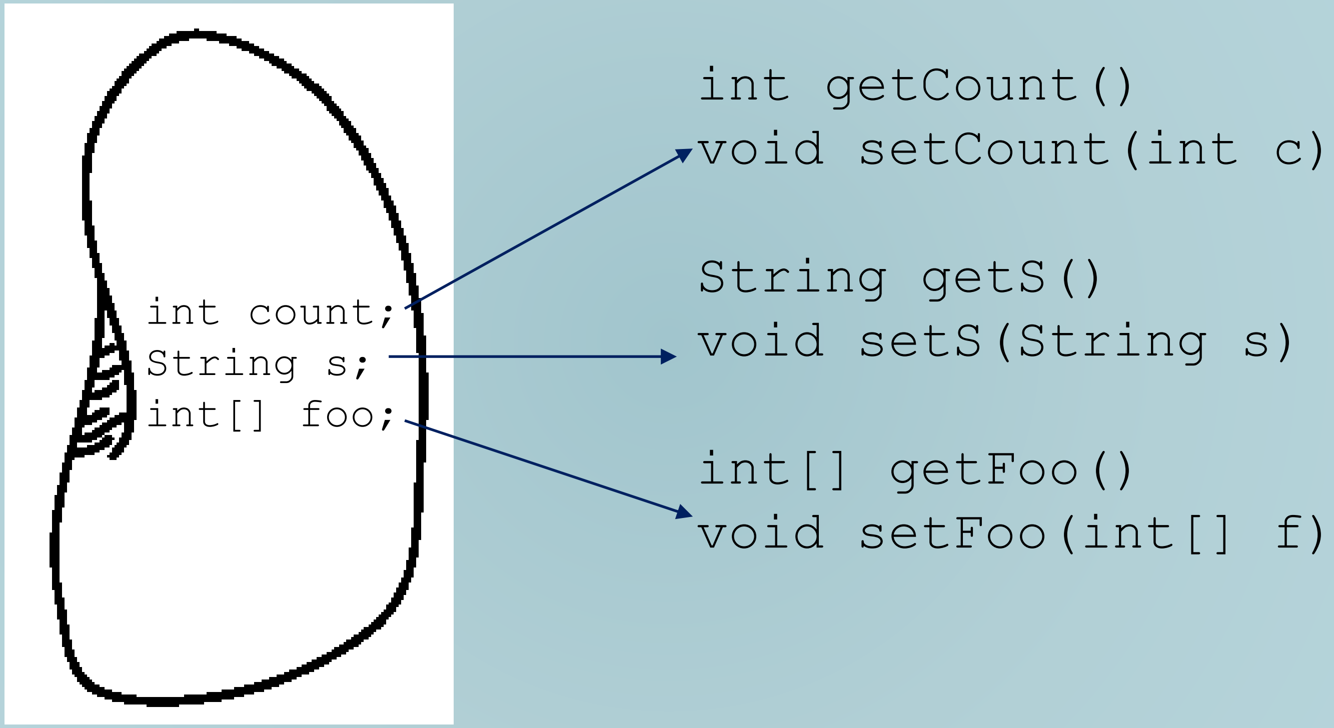
For a "foo" property, a java bean will respond to

- Type getFoo()
- void setFoo(Type foo)

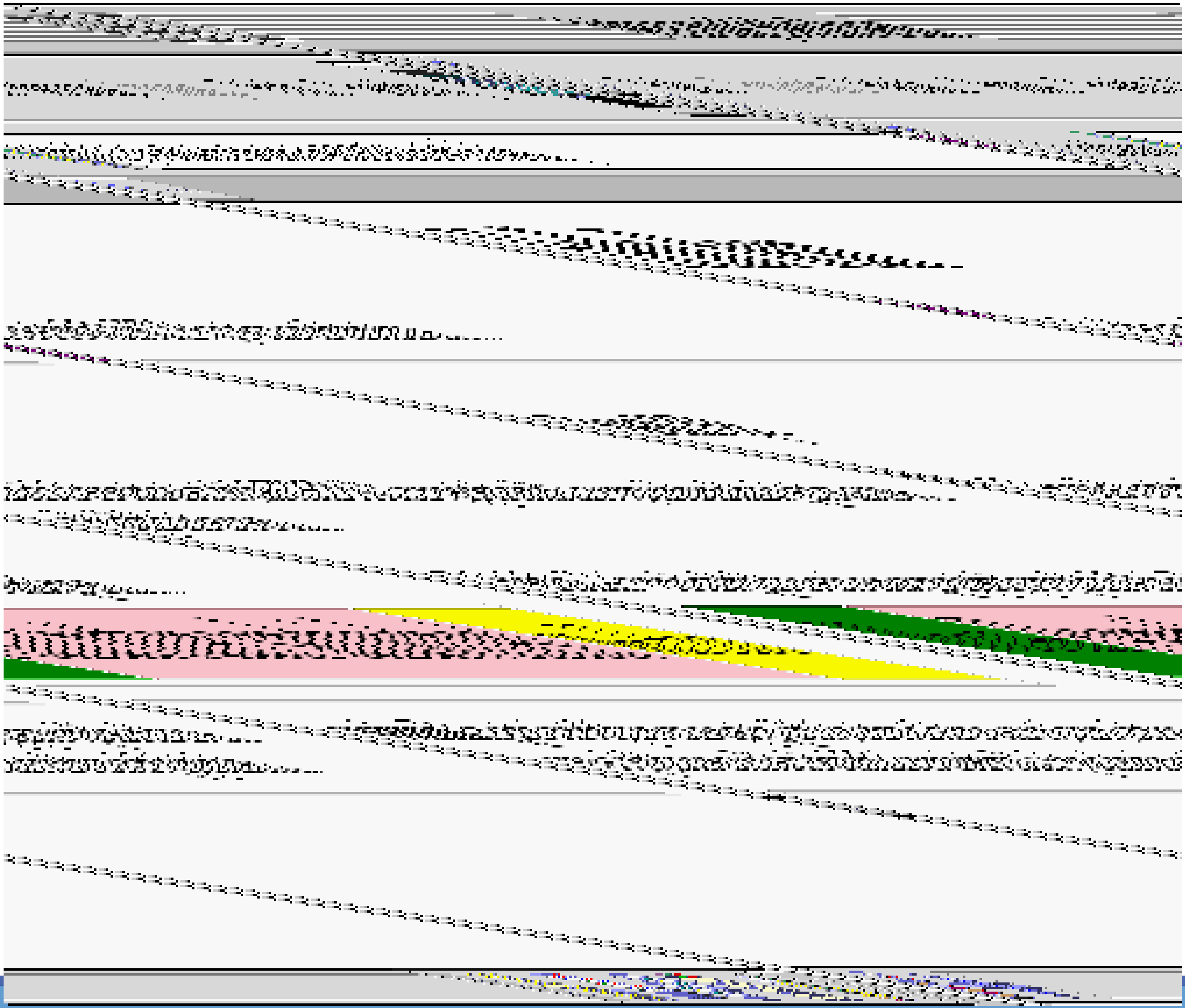
For a boolean "bar" property, a java bean will respond to

- boolean isBar()
- void setBar(boolean bar)

Java Bean



```
int getCount()
void setCount(int c)

String getS()
void setS(String s)

int[] getFoo()
void setFoo(int[] f)
```


[illegible]

```java
// MagicBean.java
/* A simple bean that contains a single
* "magic" string.
*/
public class MagicBean {
  private String magic;
  public MagicBean(String string) {
    magic = string;
  }
  public MagicBean() {
    magic = "Woo Hoo"; // default magic string
  }
  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" />

```
<!-- 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" -->



<table border=1>
<tr>
<td bgcolor=green><font size=+2>Woo</font> Hoo</td>
<td bgcolor=pink>
<font size=+3>
<td bgcolor=pink>
<font size=+3>
  <!-- the following effectively does bean.getMagic() -->

</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...");
     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

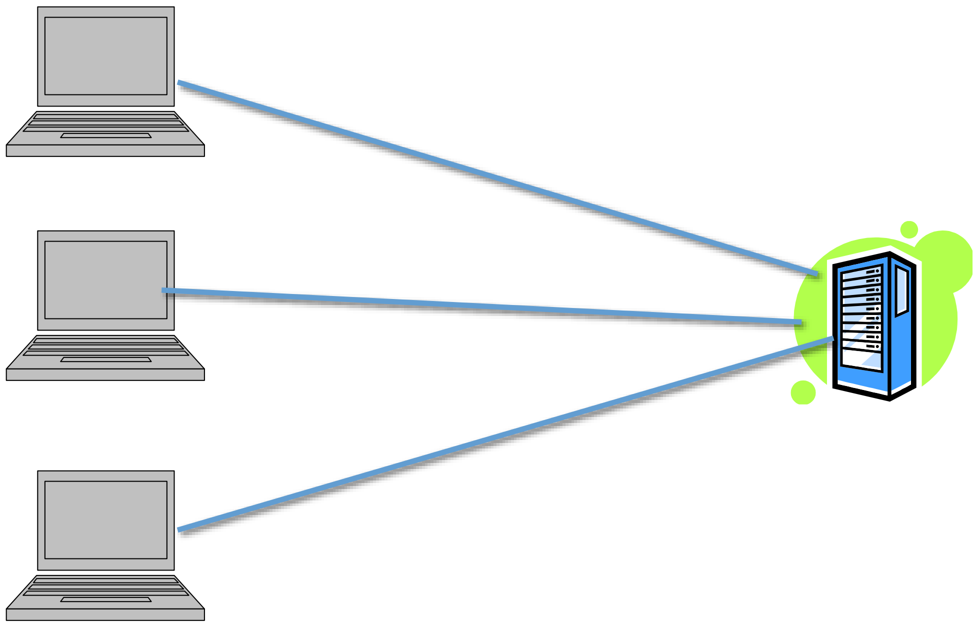
application

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

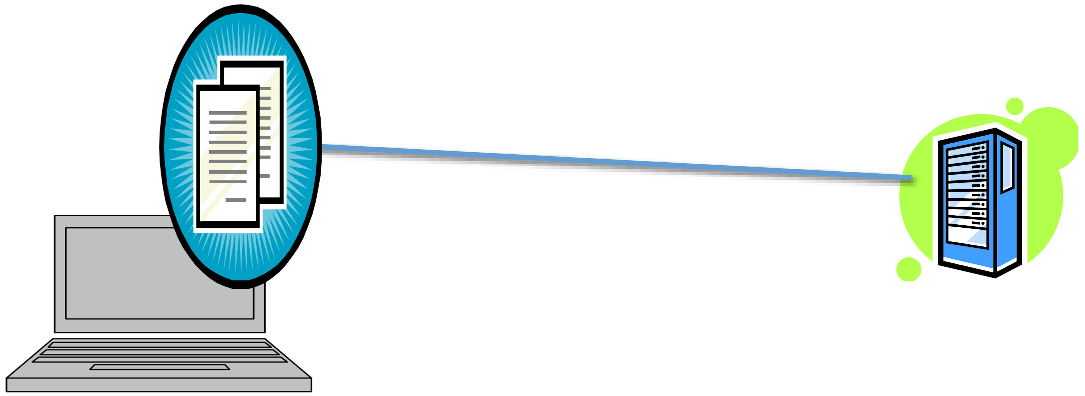
application

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

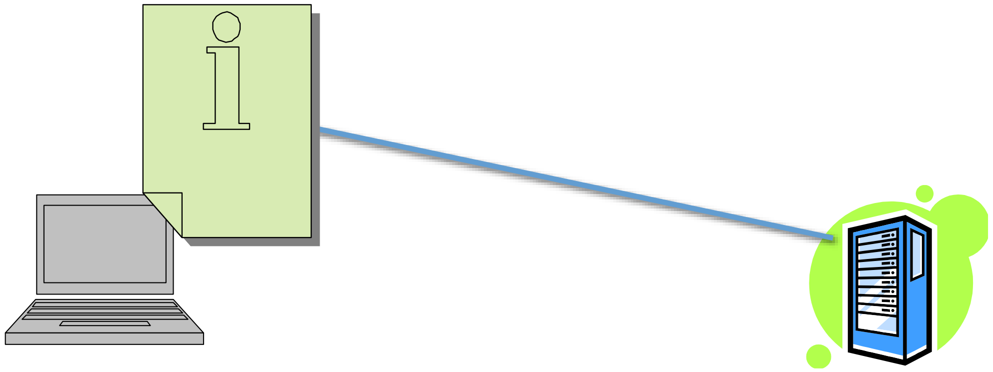
application

session

page

request

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



Scope Attributes

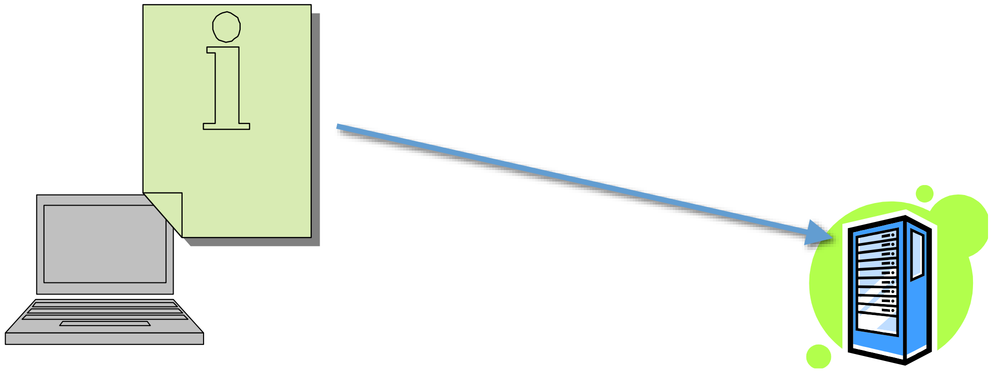
application

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 preexisting bean is used; otherwise, a new bean is created.

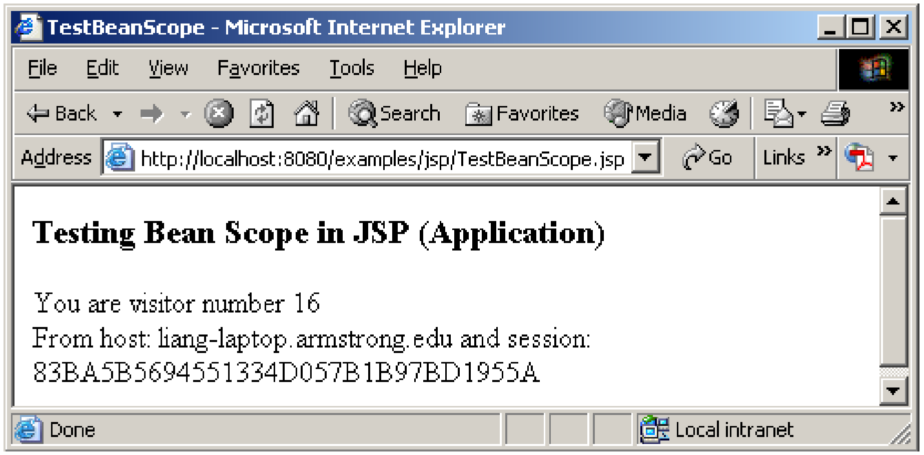
Another Syntax for Creating a Bean

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

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

 some statements

</jsp:useBean>
```

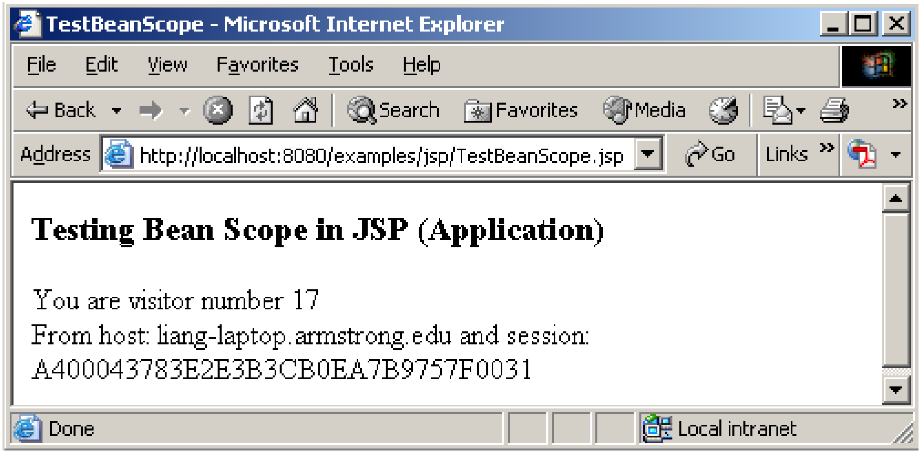
The statements are executed when the bean is created. If the bean with the same id and className already exists, the statements are not executed.

Example: Testing Bean Scope


TestBeanScope - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/examples/jsp/TestBeanScope.jsp
Go Links
Done
Local intranet

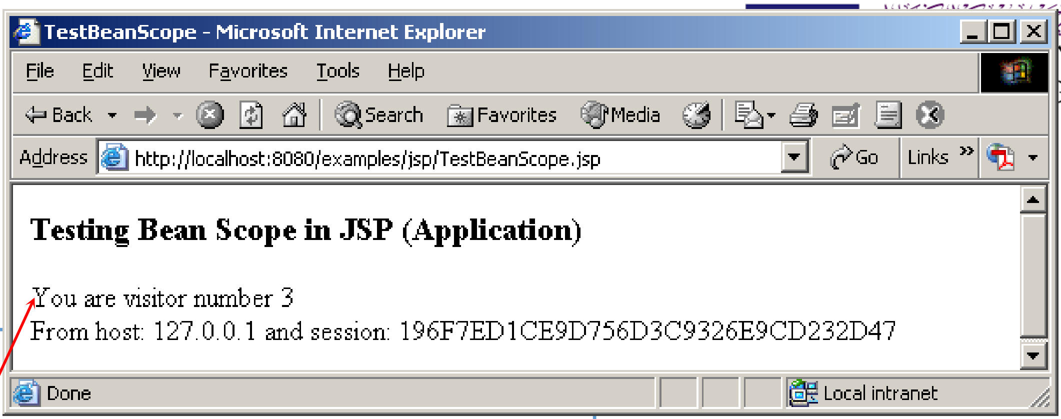
Testing Bean Scope in JSP (Application)

You are visitor number 16
From host: liang-laptop.armstrong.edu and session: 83BA5B5694551334D057B1B97BD1955A


TestBeanScope - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/examples/jsp/TestBeanScope.jsp
Go Links
Done
Local intranet

Testing Bean Scope in JSP (Application)

You are visitor number 17
From host: liang-laptop.armstrong.edu and session: A400043783E2E3B3CB0EA7B9757F0031


TestBeanScope - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/examples/jsp/TestBeanScope.jsp
Go Links
Done
Local intranet

Testing Bean Scope in JSP (Application)

You are visitor number 3
From host: 127.0.0.1 and session: 196F7ED1CE9D756D3C9326E9CD232D47

```
<!-- TestBeanScope.jsp -->
<%@ page import = "chapter40.Count" %>
<jsp:useBean id="count" scope="application" class="chapter40.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 chapter40;

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 40.3 by associating the bean properties with the input parameters.

```
<!-- ComputeLoan.jsp -->
<html>
<head>
<title>ComputeLoan Using the Loan Class</title>
</head>
<body>
<%@ page import = "chapter40.Loan" %>
<jsp:useBean id="loan" class="chapter40.Loan"></jsp:useBean>
<jsp:setProperty name="loan" property="*" />
Loan Amount: <%= loan.getLoanAmount() %><br>
Annual Interest Rate: <%= loan.getAnnualInterestRate() %><br>
Number of Years: <%= loan.getNumOfYears() %><br>
<b>Monthly Payment: <%= loan.monthlyPayment() %><br>
Total Payment: <%= loan.totalPayment() %><br></b>
</body>
</html>
```

Getting

Associating the bean properties with the input parameters.

Enterprise Java Beans

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

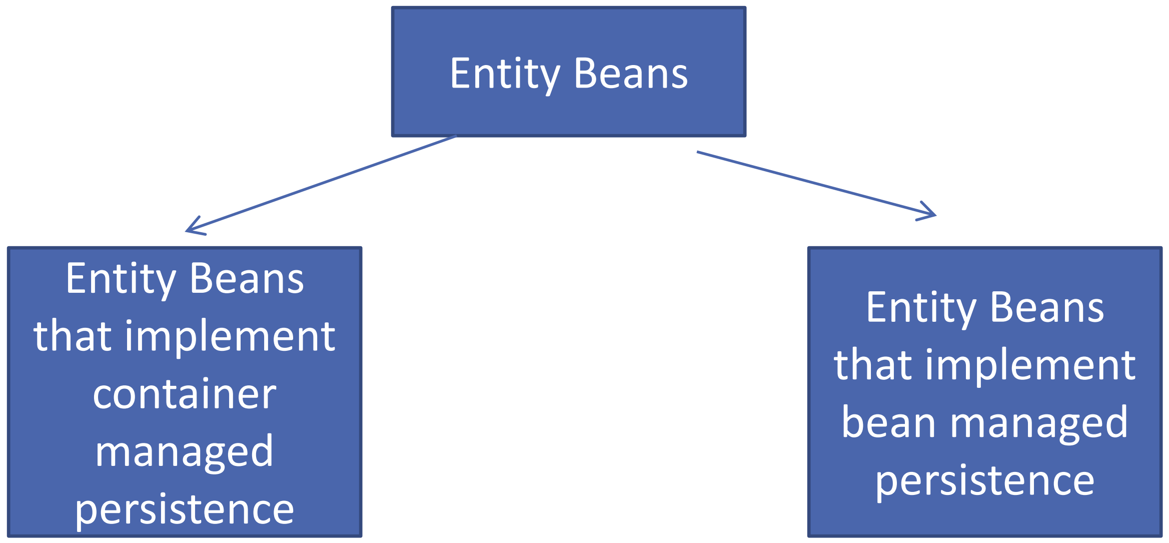
Enterprise JavaBeans technology supports both transient and persistent objects.

A transient object is called a *session bean*,

and a persistent object is called an *entity bean*.

Enterprise Java Beans

Types of Entity Beans:

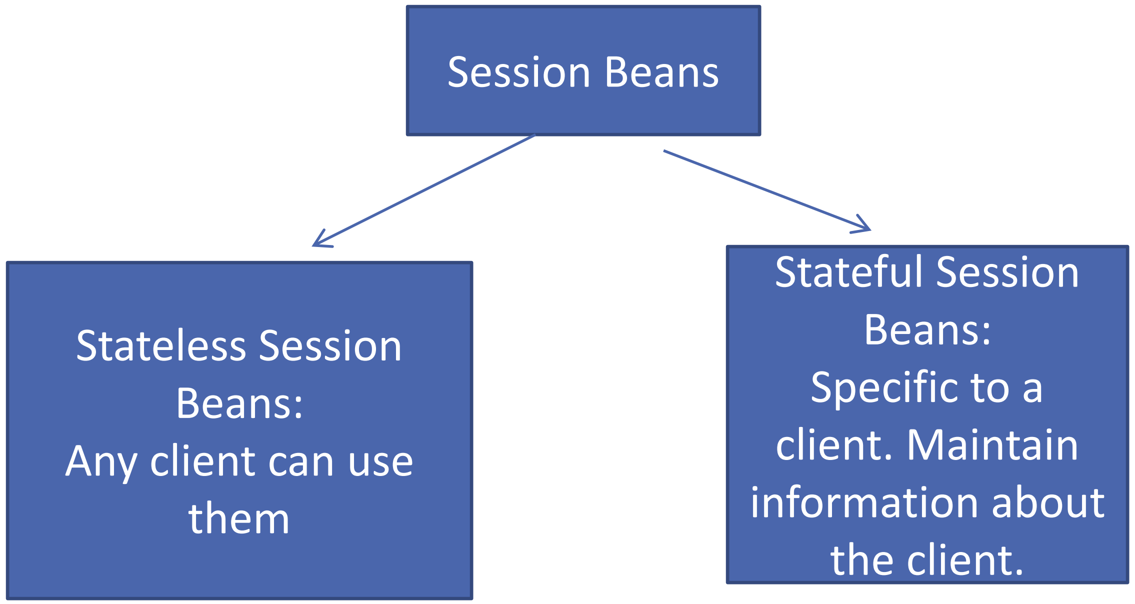

Entity Beans
Entity Beans that implement container managed persistence
Entity Beans that implement bean managed persistence

These beans are defined Independent of the database that they use

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

Enterprise Java Beans

Types of session beans:


Session Beans
Stateless Session Beans: Any client can use them
Stateful Session Beans: Specific to a client. Maintain information about the client.

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 or file and swapped out

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

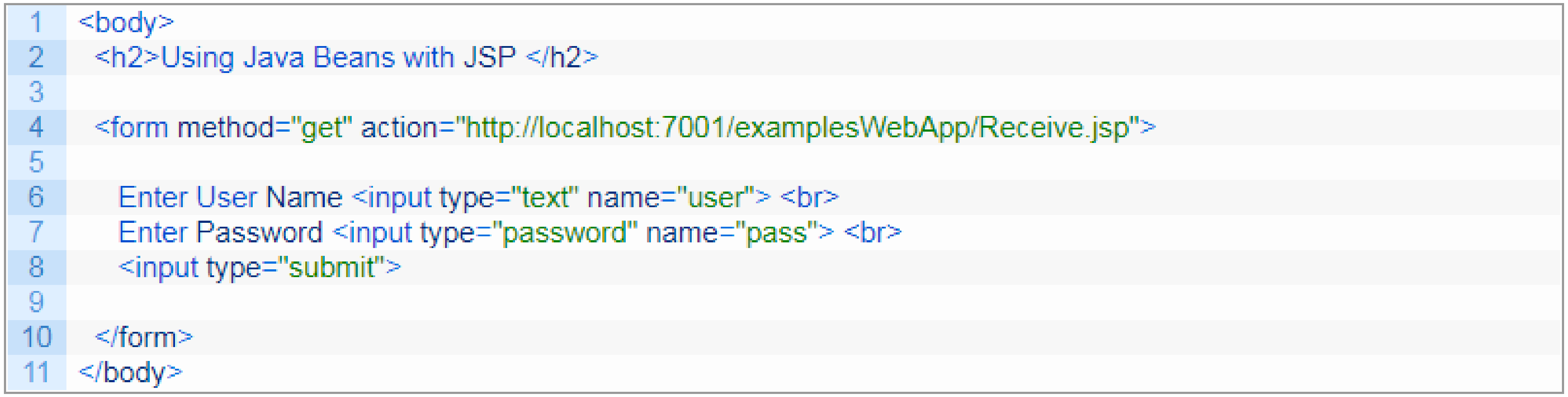
- EJB container is responsible for saving and retrieving bean’s state
- Independent of data source
- Easy to develop

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

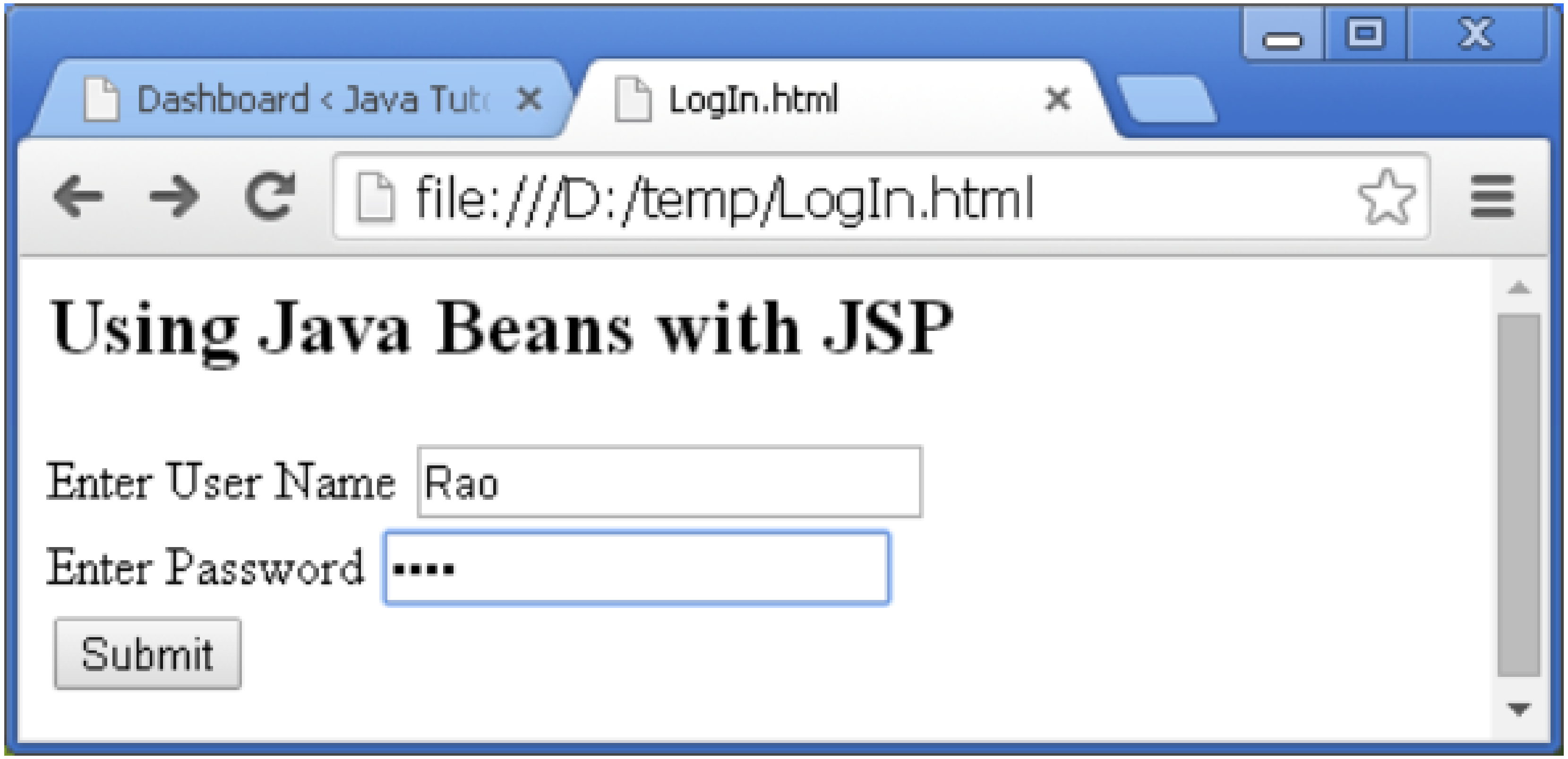
Example on JavaBean JSP

1. Client File Name: LogIn.jsp



```
<body>
  <h2>Using Java Beans with JSP </h2>

  <form method="get" action="http://localhost:7001/examplesWebApp/Receive.jsp">

     Enter User Name <input type="text" name="user"> <br>
     Enter Password <input type="password" name="pass"> <br>
     <input type="submit">

  </form>
</body>
```

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



Dashboard ‹ Java Tut

LogIn.html

file:///D:/temp/LogIn.html

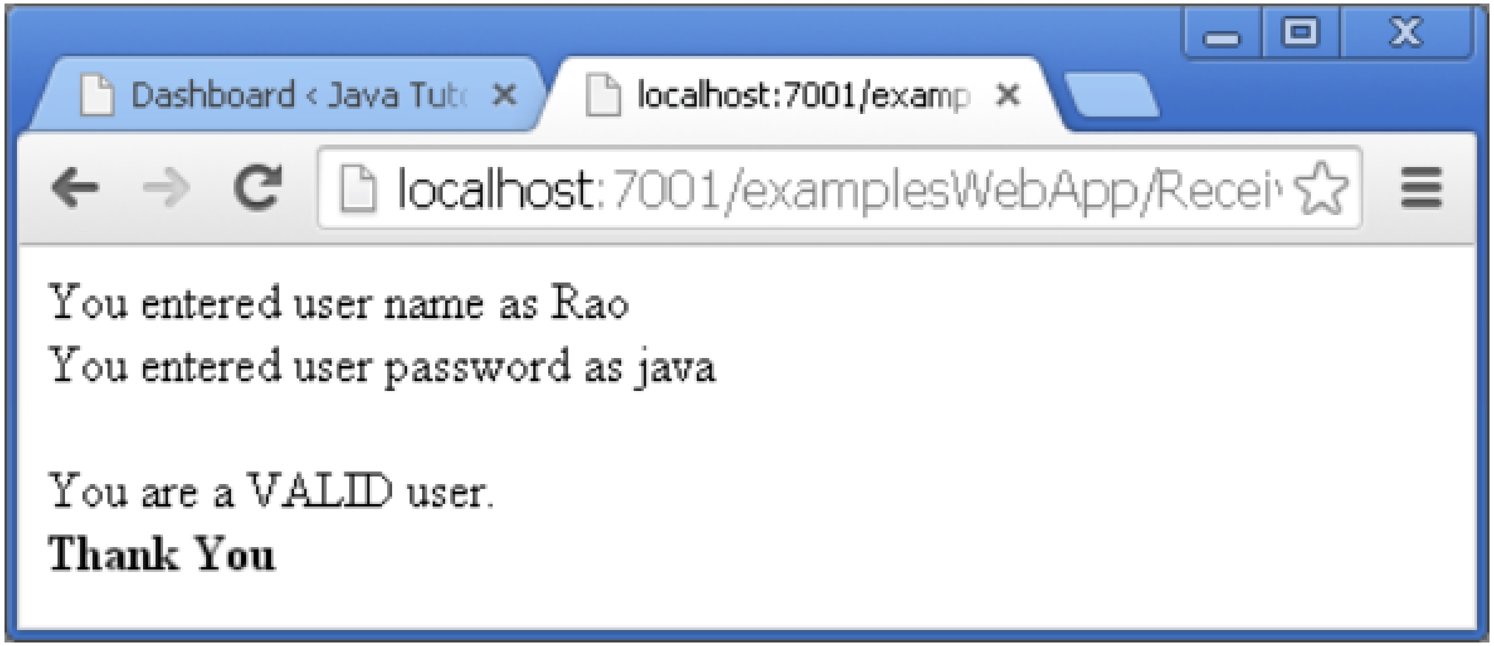
Using Java Beans with JSP

Enter User Name: Rao

Enter Password: ••••

2. Server-side File Name: Receive.jsp

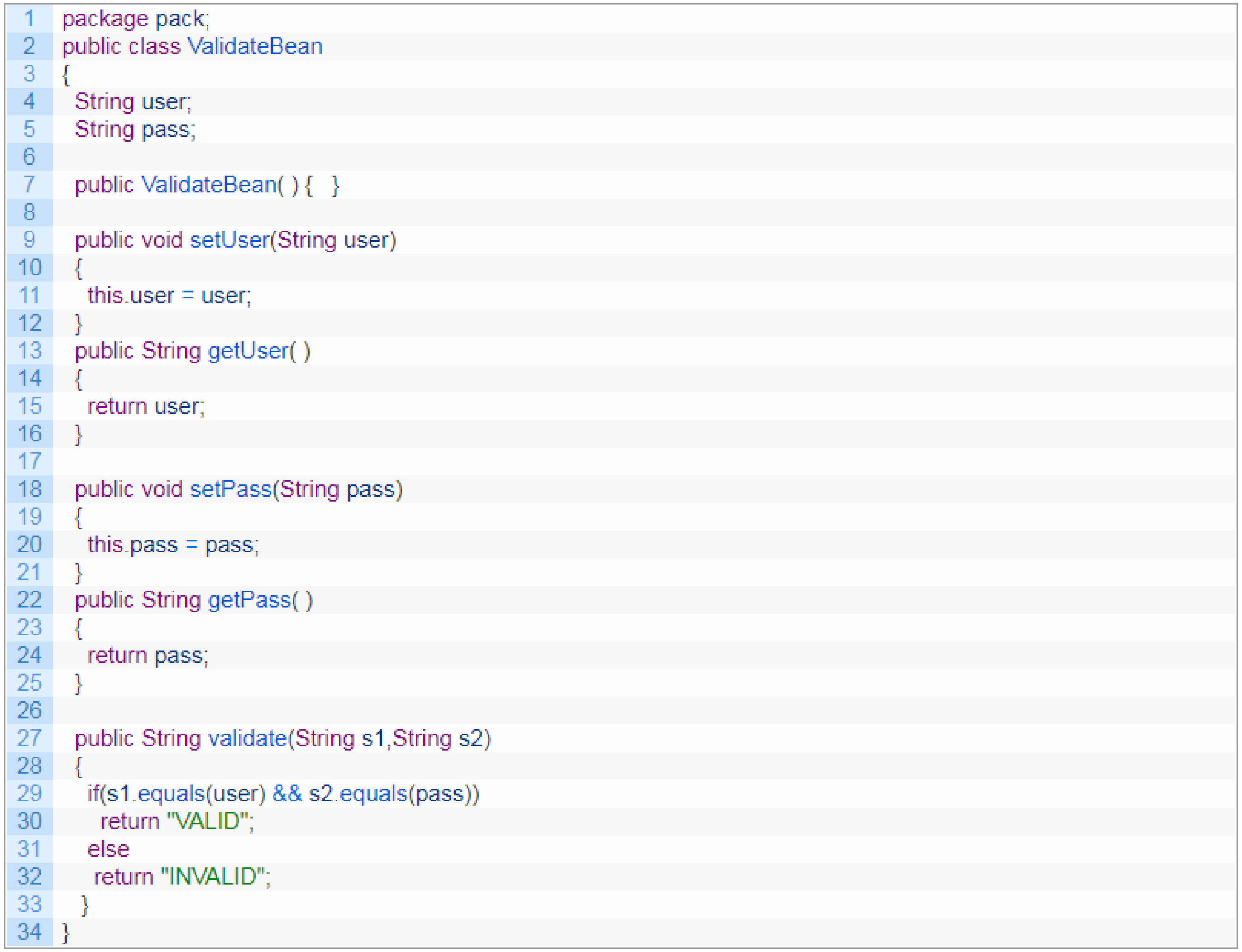
```
 <body>
  <jsp:useBean id="snr" class="pack.ValidateBean" />

  <jsp:setProperty name="snr" property="user" />
  <jsp:setProperty name="snr" property="pass" />

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

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

  <br>

  You are a <%= snr.validate("Rao", "java") %> user.  <br>

  <b>Thank You</b>
</body>
```



```
You entered user name as Rao
You entered user password as java

You are a VALID user.
Thank You
```

UNIVERSITI TEKNOLOGI MARA

3. Server-side File Name: ValidateBean.java



```java
package pack;
public class ValidateBean
{
 String user;
 String pass;

 public ValidateBean( ) {   }

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

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