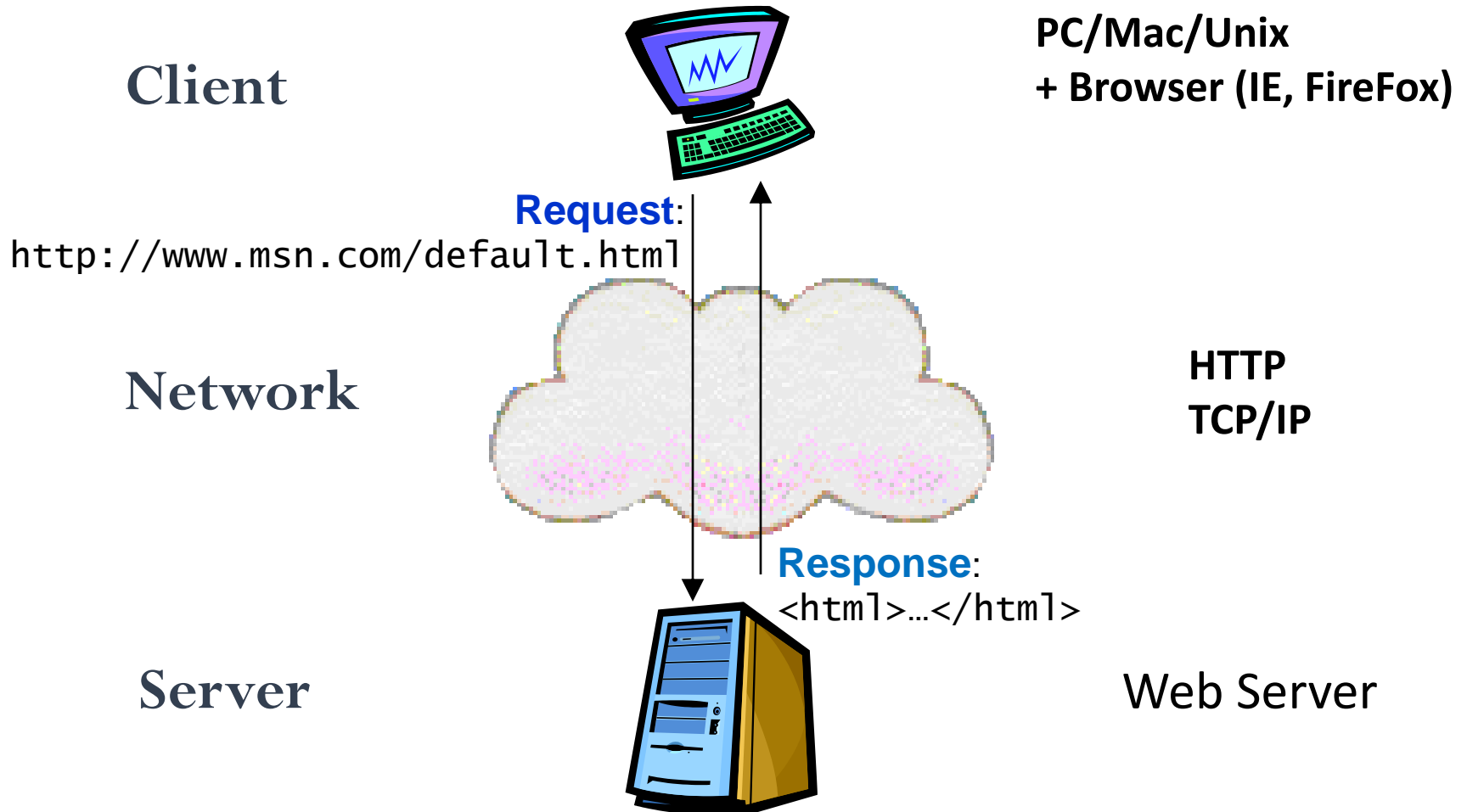


# Advanced Java

***Dr. Moumita Basu***  
***Associate Professor, CSE***

# Internet Technologies

## WWW Architecture



# What is the WWW?

- Uses a client-server model
- Main presentation language is HTML

# Client-Server Model

Two processes (possibly networked):

- The client
  - Sends requests to the server
  - Blocks until reply is received
- The server
  - Processes requests from clients
  - Never blocks
  - Can reply to several clients simultaneously

# Web Server

- Server is a software which serves the request.
- A web server takes a client request and gives something back to the client..
- A web browser lets a user request a resource. The web server gets the request, find the resource and return something to user. Resource is an html page or sound file, picture or a .pdf file.

# Web Client

- A web client let the user(through browser application) request something on the server and shows the user the result of the request.
- The client request contains name and address(url) of the resource.
- Browser is a piece of software that knows how to communicate with the server. It also interpret the html code and render the web page .

# HTTP

- If you're a web server you speak HTTP.
- HTTP is the protocol(set of rules) client and server use on web to communicate.
- HTTP runs on top of TCP/IP.
- TCP is responsible for making sure the file send from one network to another ends up as a complete file at destination, even though the file is split into chunks (packets)when sent.
- IP is the underlying protocol that routes the packets from one host to another on their way to destination.

# Web Technologies

- HTTP / HTTPS (URL, GET/POST)
- Client-side:
  - HTML / XHTML (Extensible HyperText Markup Language)
  - JavaScript / VBScript (client-side scripting)
- Server-side:
  - PHP
  - Python
  - J2EE
  - ASP.NET (next generation of ASP)

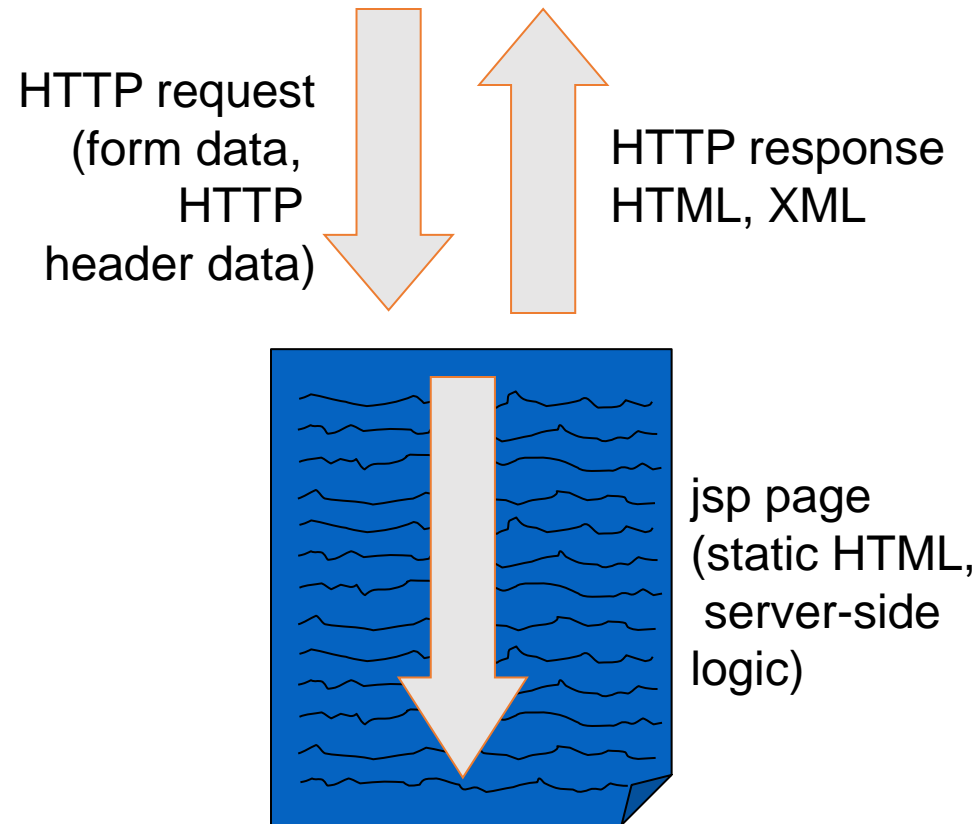


# What are different flavours of java?

- JSE (Standard Edition) [Core java, standalone java programs]
- JEE (Enterprise Edition) [ Advanced java, web based programming]
- JME (Micro Edition) [ Close to assembly language coding, microprocessors are burnt to write JME, meant for embedded devices.]

# Server-Side Code

- What is server-side code?
  - Software that runs on the server, not the client
  - Receives input from
    - URL parameters
    - HTML form data
  - Can access server-side databases, e-mail servers, files, mainframes, etc.
  - Dynamically builds a custom HTML response for a client



# HTML (Hypertext MarkUP Language)

- HTML stands for HyperText Markup Language.
- Markup Language is a computer language that consists of easily understood keywords, names, or tags that help format the overall view of a page and the data it contains. Some examples of a markup language are **BBC, HTML, SGML, and XML**.
- HTML is used to create web pages and web applications.
- HTML is widely used language on the web.
- We can create a static website by HTML only.
- Technically, HTML is a Markup language rather than a programming language.

# MarkUP Language (HTML, CSS or XML)

- Markup languages prepare a structure for the data or prepare the look or design of a page
- These are ***presentational*** languages and it doesn't include any kind of logic or algorithm
- It tells the browser how to structure data for a specific page, layout, headings, title, table and all or styling a page in a particular way
- It involves formatting data or it controls the presentation of data
- These languages are most widely used to design a website.

# Scripting Language

- Scripting languages are basically the subcategory of programming languages
- Scripting languages need to be interpreted (Scanning the code line by line, not like compiler in one go) instead of compiled

# HTML (Hypertext MarkUP Language)

- HTML is the *lingua franca* for publishing hypertext on the World Wide Web
- Define tags <html><body> <head>....etc
- Allow embedding other scripting languages to manipulate design layout, text and graphics
- Platform independent

## HEML TAGS

1. Html tags are used to markup html element.
2. Html tags are surrounded by <and> Characters these characters are called as angler brackets.
3. Almost all Html tags come in pairs. That is tags contain a starting tag and ending tags.
4. The text in b/w the starting tag and ending tag is called as element content.
5. Html tags are free defined tags.
6. Html tags are not case sensitive. That the upper and lower document is html document.
7. The html tag <html> in a document represents. That the document is html document.
8. The entire html document must be written starting html tag <html> and ending html </html>.
9. Html document is divided into two sections.

# HTML

## **(I) Head section:**

This section is used to provide general information about the html doc. This section is represented by <head>.

Ex: Title, Meta etc.

## **(II) Body section:**

This section is used to display text and images on the Browser. This section is represented by <body>.

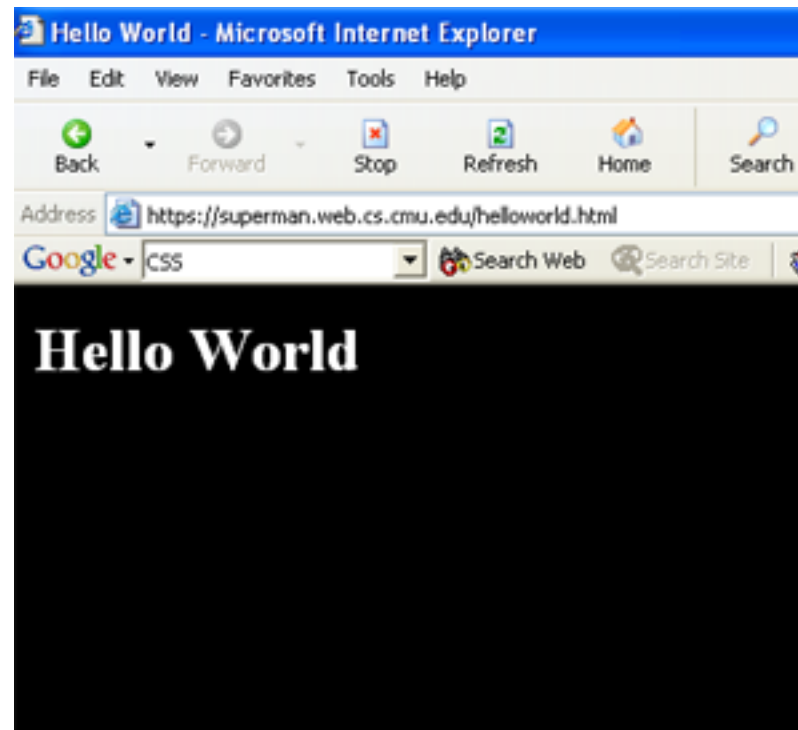


# HTML (Hypertext Markup Language)

- Example HTML code:

```
<HTML>
<head>
<title>Hello World</title>
</head>
<body bgcolor = “#000000”>
<font color = “#ffffff”>
<H1>Hello World</H1>
</font>
</body>
</HTML>
```

# HTML (Hypertext Markup Language)



# Html comments:

Comments are used to make the code name readable or they are used to explain the code.

HTML comments begins with `<!--line one comment-->`

EX :`<!--line one comment -- >`

`<!--line one comment`

`line two comment`

`.`

`.`

`.`

`N no. of line comments -- >`

# Attributes

1. Attributes are used to provide additional information about the HTML elements tags.
2. The attributes must be specified in starting tags.
3. The attributes always come in name-value pairs.  
attribute name = attribute value
4. The attribute value can be enclosed with in single quotes are double quotes.
5. Every HTML tag can contain attributes.

Note: Specifying the attributes for a tag is optional.

# Body tag attributes:

- 1. **Bgcolor**: This attribute specifies the background color to be displayed for an HTML document.
- 2. **Text** : This attribute specifies the color of the text to be displayed on the HTML document. Color can be specified (on the HTML document) for two formats.
  - a. Specified the color value directly.
  - b. We can specify the color by combining Red, Green and Blue. The format should be #rrggbb. The color combination is 6 digit Hexadecimal numbers.
- 3. **Background**: attribute specifies an image as the background to the HTML document.
- This
- `<body bgcolor = "#ffff00" text = "#ff0000">`
- Good Afternoon
- `</body>`
- Note: Specifying the attributes is optional, we can specify any number of attributes and they can be specified in any order.

# Table tag

**<Table>**: This tag is used to display a table in a html document

## **Attributes of <table>:**

1. **Border**: This attribute specifies the thickness of the border to be displayed. By default are zero pixels.
2. **Border color**: This attribute specifies the color of the border to be displayed.
3. **Width**: This attribute specifies the width of the table.
4. **Height**: This attribute specifies the height of the table.
5. **Align**: This attribute can be used to align a table in a web page.
6. **Bgcolor**: This attribute specifies a bgcolor for the table.
7. **Background**: This attribute is used to display an image the background to be table.
8. **Cell padding**: This attribute specifies the amount of space between cell content and cell border.
9. **Cell spacing**: This attribute specifies the space between the cells and b/w the cell border and table border.

# Sub tags of <table>

1. **<tr>(table row)**: This tag is used to create a row in a table. The no. of <tr> will indicate the no. of row in a table.
2. **<td>(table data)**: This tag is used to create a column(cell) in a particular row. The no. of <td>'s will indicate no. of columns in a particular row.
3. **<th>(table heading)**: This tag is used to create a column in a particular row similar to <td> but <th> will display the text in bold face and the text will be centered.
4. **<caption>**: This tag is used to display a heading to a table.

# Attributes of sub tags:

1. **Row span:** This attribute is used to span (merge) the cells vertically.
2. **Colspan:** This attribute is used to span (merge) the cells horizontally.
3. **Valign:** This attribute is used to align the text vertically.

Valign = “top | middle | bottom” by the default it is vertically aligned to middle. In this sub tags we can also use align, bgcolor, and background attributes.



# Attributes of<img>

1. **Src(source):** This attribute specifies of the URL(Address of the image).
2. **Border:** This attribute specifies of the thickness of the border around the image. By default thickness is 'zero' pixels.
3. **Width:** This attribute specifies of the width of the an image to be displayed in a html document.
4. **Height:** This attribute specifies of the Height of the image to be displayed in a html document.

# Attributes of<img>

5. **Alt(alternate text)**: This attribute specifies a text to be displayed on the html document when an image is not loaded.

6. **Hspace(Horizontal Space)**: This attribute specifies the amount of space empty space to be displayed on the left and right sides of the image.

7. **VSpace(vertical space)**: This attribute specifies the amount of empty space to be displayed on the top and bottom of the image.

**Ex:**`<img src = "smile.jpg" border = "2" width = "100" height = "100" alt = "smile logo" hspace = "50" vspace = "30"/>`

**Note:** An <img> is an empty tag and (source) "src" is a mandatory attribute to display an image.

# Attributes of <a>:

**<a>**: This tag is use to create link b/w the html documents are any other web resource.

## Attributes of <a>:

1. **Href(hyperlinking reference)**: This attribute specifies the URL(address) of the resource to which the hyperlink as the link reference.

2. **Target**: This attribute will specifies were to open the linked document.

Target = “\_self” | “\_blank” (\_under scor) the default value \_self.

**Ex:** <a href = “image.html” target = “\_blank”>click here</a>

# Formatting tags

**<b>**: This tag is use to display a text in Bold face.

**<b>Bold</b>**

**<strong>**: This tag is similar to <b> used to display the text in Bold face.

**<strong>Bold</strong>**

**<i>**: This tag is use to emphasize a text by display in Italics format.

**<i>Italics</i>**

**<em>**: This tag is use to emphasize a text by display in Italics format.

**<em>Italics</em>**

**<tt>**: This tag is use to display a text in Teletype font. That is typewriter font.

**<tt>Teletype</tt>**

**<s>**: This tag is used to display a Strike the text.

**<s>Strike</s>**

# Formatting tags

**<strike>**: This tag is similar to <s> use to display a Strike out text.

**<strike>**Strike**</strike>**

**<del>**: This tag is use to Delete a text by Striking it out.

**<del>**Strike**</del>**

**<ins>**: This tag is use to Insert a line below the text.

**<ins>**underline**</ins>**

**<u>**: This tag is use to underline a text.

**<u>**underline**</u>**

# Heading tags

These tags are used to perform font changes display the text in boldface and the heading tags by default align the text tag left side. These are six levels of heading tags. H1, H2, H3, H4, H5 and H6. H1 is the biggest in heading and H6 is the smallest in heading and H6 is the smallest heading.

We can change the alignment of the headings by using align attributes. Ex: align = “left | center | right”

<H1 align = “left”>Heading1</H1>

<H2 align = “left”>Heading2</H2>

<H3 align = “center”>Heading3</H3>

<H4 align = “center”>Heading4</H4>

<H5 align = “right”>Heading5</H5>

<H6 align = “right”>Heading6</H6>

# Formatting tags

**<sup>**: This tag is use to display a text as a Superscripted. That the text will be displayed above the normal text.

a<sup>2</sup>+b<sup>2</sup>

**<sub>**: This tag is use to display a text as a Subscripted. That is the text will be displayed below the normal text.

H<sub>2</sub>SO<sub>4</sub>

**<big>**: This tag is use display a text solidly Bigger then the normal text.

<big>text</big>

**<small>**: This tag is to display a text solidly Smaller then the normal text.

<small>text</small>

**<center>**: This tag is to display a text in the Center of the webpage(html document).

<center>Welcome</center>

# <pre>:(preformatted text)

This tag is used to display a text in preformatted manner. The browser preserves all the white spaces as it is.

Ex:<pre>

To,

The manager,

HDFC bank,

Nellore,

Sub: Regarding loan to buy a house.

Respected sir/madam,

.....

<\pre>



## **<p> tag**

**<p>**: This tag is used to display a paragraph. The **<p>** will add paragraph break (inserts an empty line before and after the paragraph). The paragraphs are by default to left side. We can change the alignment by using the alignment attribute.

- Ex: align = "left | center | right"
- **<p align = "left">**this is a paragraph**</p>**
- **<p align = "center">**This is a paragraph**</p>**
- **<p align = "right">**This is a paragraph**</p>**

**<hr>**: This tag is used to display a horizontal line in a html document. Using the horizontal rule we can divide the web page into multiple sections.

# Attributes of <hr>:

1. **Width:** This attribute specifies the length of the Horizontal rule to be displayed in a web page. The width can be specified either in pixels or percentages.

Ex:<hr width = "300"/> or <hr width = "75%"/>

The width can be specified either in pixels or percentage.

2. **Color:** This attribute specifies the color of the horizontal rule to be displayed in a web page.

Ex:<hr color = "red"/>

3. **Size:** This attribute specifies the thickness of the horizontal rule to be displayed.

Ex:<hr size "10"/>

4. **align:** We can change the alignment of the horizontal rule by using this align attribute. By default the horizontal rules are aligned to center.

Ex :< hr align = "center"/>

<hr width = "75%" align = "center" color = "blue" size = "10"/>

**Note:** hr is an empty tag.

# List

In html document we can present the data in the form of list. There are 3 types of lists.

1. Ordered List
2. Unordered List
3. Definition List

# Ordered List

A list is said to be an ordered list if the items are displayed by using either digits or by Alphabets or Roman numbers. By default, digits are used to display order list. We can change the display by using “type” attribute.

Type = “I | a | A | i | II|”

**Ex:** <ol type= “I”>

<li>Hyderabad</li>

<li>Mumbai</li>

<li>Chennai</li>

</ol>

# Unordered List

If a list is displayed with the help of some graphical symbols then it is called as unordered list. We can change the display by using “Type” attribute.

Type = “disc | square | circle”

**Ex:**

```
<ul>
```

```
<li>Hyderabad</li>
```

```
<li>Mumbai</li>
```

```
<li>Chennai</li>
```

```
</ul>
```

# Data list with Data definition

3) **Definition List:** A list is said to be a definition list if we provide a definition or description to every item without any bullet or Number

**Ex:**

```
<dl>  
<dt>Hyderabad</dt>  
<dd>Biryani</dd>  
<dt>Chennai</dt>  
<dd>somber</dd>  
<dt>Mumbai</dt>  
<dd>bomb</dd>  
</dl>
```

We can place a list inside another list that is nesting of lists is possible.

# Nested List

```
<ul>
```

```
<li>Hyderabad</li>
```

```
<ol>
```

```
<li>Ameerpet</li>
```

```
<li>Hi-tech</li>
```

```
<li>Birla temple</li>
```

```
</ol>
```

```
<li>Mumbai</li>
```

```
<li>Chennai</li>
```

```
</ul>
```

# <Font> Tag

**<Font>**: This tag can be used to apply font changes, change the text color or face or size.

## **Attributes of <font>:**

1. **Color**: This attribute specifies the color of the text to be displayed.

**Ex:** <font color = "red"> Hello </font>

2. **Size**: This attribute specifies the font size to be displayed.

**Ex:** <font size = "10">Hello</font>

3. **Face**: face will specify the place of the text to be display.

**Ex:** <font face = "Arial">Hello </font>

<font color = "red" size = "10" face = "Arial"> Hello</font>

**Note:** Font tag is dependent tag. That is <font> will not apply any changes without the attributes.



# <Marquee>:

**<Marquee>:** This tag use to display a scrolling text on a web page.

## **Attributes of <Marquee>:**

1. **Behavior:** This attribute specifies the behavior of the scrolling text in a web page.

**Ex:**

Behavior = “scroll | slide | alternate”(the default behavior is scroll)

2. **Direction:** This attribute specifies the direction in which the text will be scrolled.  
By default the text scrolls towards left side.

3. **Bgcolor:** This attribute will display bgcolor to the scrolling text.

4. **Scroll amount:** This attribute specifies the speed of the scrolling text.

**Ex:**

```
<marquee behavior = “scroll” direction = “right” bgcolor = “red” scrollamount = “45”>flash news</marquee>
```

# FORM

- **Forms:** Form is an area which contains some input elements where the user can provide the information. Forms are used to submit the data to the server. The html document can have any no. of forms. To create a form in html document we use `<form>`.

# Attributes of <form>

1. **Action:** This attribute specifies the URL(address) of the server where the form data has to be submitted.
2. **Method:** This attribute defines how to submit the form data to the server. The most widely values are 'get and post'. If the method value is 'get' then the form data will be submitted to the server along with the URL. The form data will be display in the address bar and hence doesn't provide security to the form data. If the method value is 'post' then the form data will be submitted to the server separately from the URL. It won't display the from data in the address bar. There by provides security to the form data. Using 'get' we can send limited data only but using 'post' we can sent unlimited data. The default method value is 'get'.
3. **Name:** This attribute is used to assign a name to a form for identification

# Sub tags of form tag:

**<input>**: This tag is used to create an input element where the user can enter some information.

## **Attributes of <input>**:

1. **Type**: This attribute will decide the type of input element to be displayed.

**Ex**: type: text, password, radio, checkbox, submit, reset, hidden (internal all pages displayed), button.

2. **Name**: This attribute is used to assign a name to the input element.

3. **Size**: This attribute specified the length of the input field.

4. **Value**: This attribute is used to specify the default value or initial value to an input element.

5. **Max length**: This attribute specify the maximum number of characters to be entered into the input element.

**<Select>**: This tag will provide the user a list of options from which the user can select one option.

**<Text area>**: This tag allows the user to enter the data in multiple lines the size of the text area can be specified by using rows and cols attribute.

## <frame>

- This tag represents an individual frame within a frameset. It divides the web page into different parts where each part contains html, JPG or GIF file, It divides the screen row-wise and columnwise

1. **Src(source)**: This attribute specifies the URL(address) of the html document to be displayed within the frameset.

2. **Scrolling**: This attribute will decide frame will contain a scrolling bar or not.

Scrolling = “auto | yes | no”

The default scrolling value is also “auto”.

3. **Name**: This attribute is used to assign a name to a frame.

4. **No resize**: It is a flag that indicates that the frame can't be resized.

# Example

```
<html>  
<frameset rows = "30%,*" border = "10">  
<frameset src = "welcome.html" noresize/>  
<frameset cols = "100,*">  
<frame src = "sample.html"/>  
<frame src = "image.html"/>  
</frameset>  
</frameset>
```

**Note:** The frameset tag must be written outside the <body>.



# JSP(Java Server Pages)

- Java Server Pages are HTML pages embedded with snippets of Java code.



# Advantages of JSP

- Separates application logic and Web page design, reducing the complexity of Web site development and making the site easier to maintain.
- JSP technology is Java-based, it is platform-independent. JSPs can run on any nearly any Web application server. JSPs can be developed on any platform and viewed by any browser because the output of a compiled JSP page is HTML.
- JSPs support both embedded JavaScript and tags. JavaScript is typically used to add page-level functionality to the JSP.
- No need to create XML file
- No need to compile separately like Servlet
- JSP support some implicit objects. No need to create them explicitly.
- JNDI architecture is not required to know.
- JSPs are translated into [servlets](#) at runtime

# JSP Elements

- Directive elements

provide information to the JSP container about the page.

- Scripting elements

the elements in the page that include the Java code

- Action elements

These elements are also known as **Standard Actions**. Standard actions are defined by the JSP specification

# Scripting Elements

- The scripting elements are the elements in the page that include the Java code. There are three sub forms of this element: declarations, scriptlets, and expressions. Their forms are:
  - `<%! declaration %>`
  - `<% scriptlet code %>`
  - `<%= expression %>`

# Scripting Elements

- `<%! declaration %>`
- This tag is used to declare global variables
- `<% scriptlet code %>`
- This tag is used to declare local variables, as well as all java codes, are written within this
- `<%= expression %>`
- This tag is used to print the value of a variable or to evaluate some arithmetic expression

# Directive Element

## Directive

## Description

`<% @ page ... %>`  
as

Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

`<% @ page attribute="value" %>`

`<% @ include ... %>`  
phase.

Includes a file during the translation

`<% @ include file="url" >`

# Directive Element

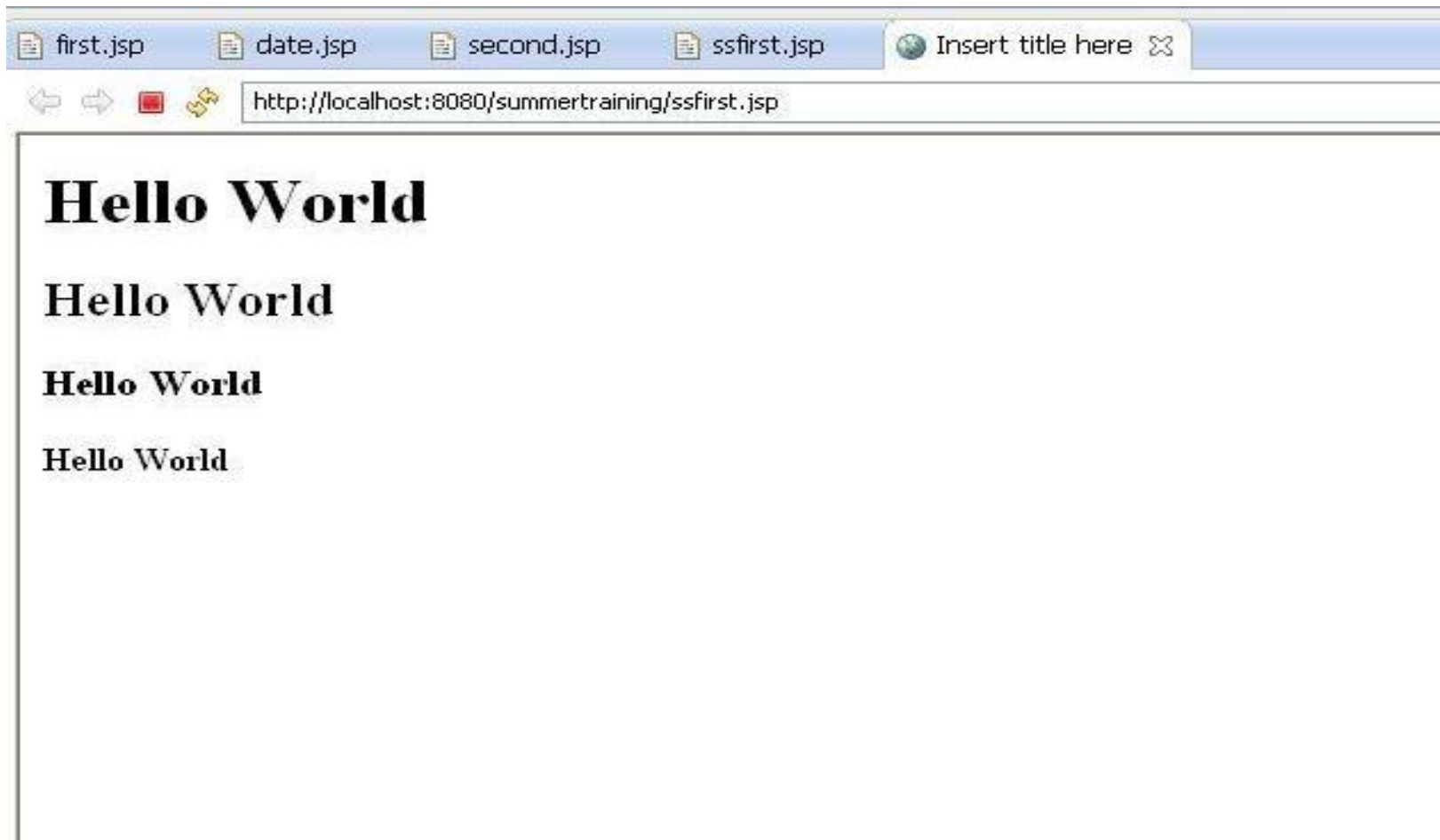
- Include directive
- `<%@ include file="URL" %>`--This tag is used to include some html or jsp page within a jsp page
- Example: Suppose header and footer part of the web page is same and made in an html or jsp page then this pages are called from every web page by using include directive
- Import directive is used to import packages within jsp page.

`<%@ page import= "java.sql.*" %>`

# Scripting Elements: Example 1

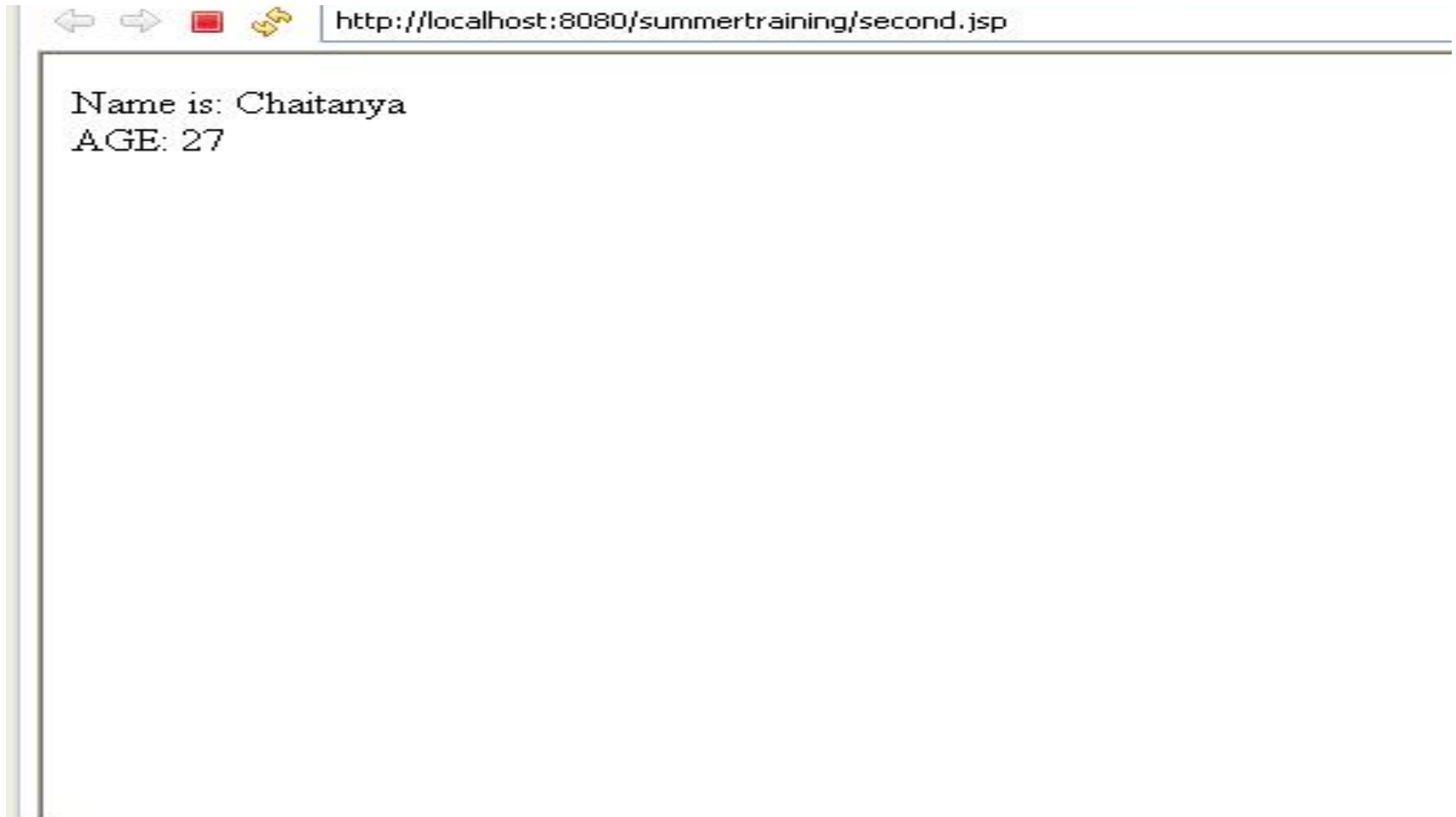
```
<html>
<head>
<title>Hello World - test the J2EE SDK installation
</title>
</head>
<body>
<%
for (int i = 1; i < 5; i++)
{
%>
<h<%=i%>>Hello World</h<%=i%>>
<%
}
%>
</body>
</html>
```

# PROBLEM 1





# PROBLEM 2



# Scripting Elements: Example 2

```
<html>
  <head>
    <title>Declaration tag Example1</title>
  </head>
  <body>
    <%! String name="Chaitanya"; %>
    <%! int age=27; %>
    <%= "Name is: "+ name %><br>
    <%= "AGE: "+ age %>
  </body>
</html>
```

# PROBLEM 3



# Directive Elements Example

Let us define following two files (a)date.jsp and (b) main.jsp as follows:

## **date.jsp file:**

```
<p>
```

```
Today's date: <%= (new java.util.Date()).toString()%></p>
```

## **main.jsp file:**

```
<html>
```

```
<body>
```

```
<center><h2>The include action Example</h2>
```

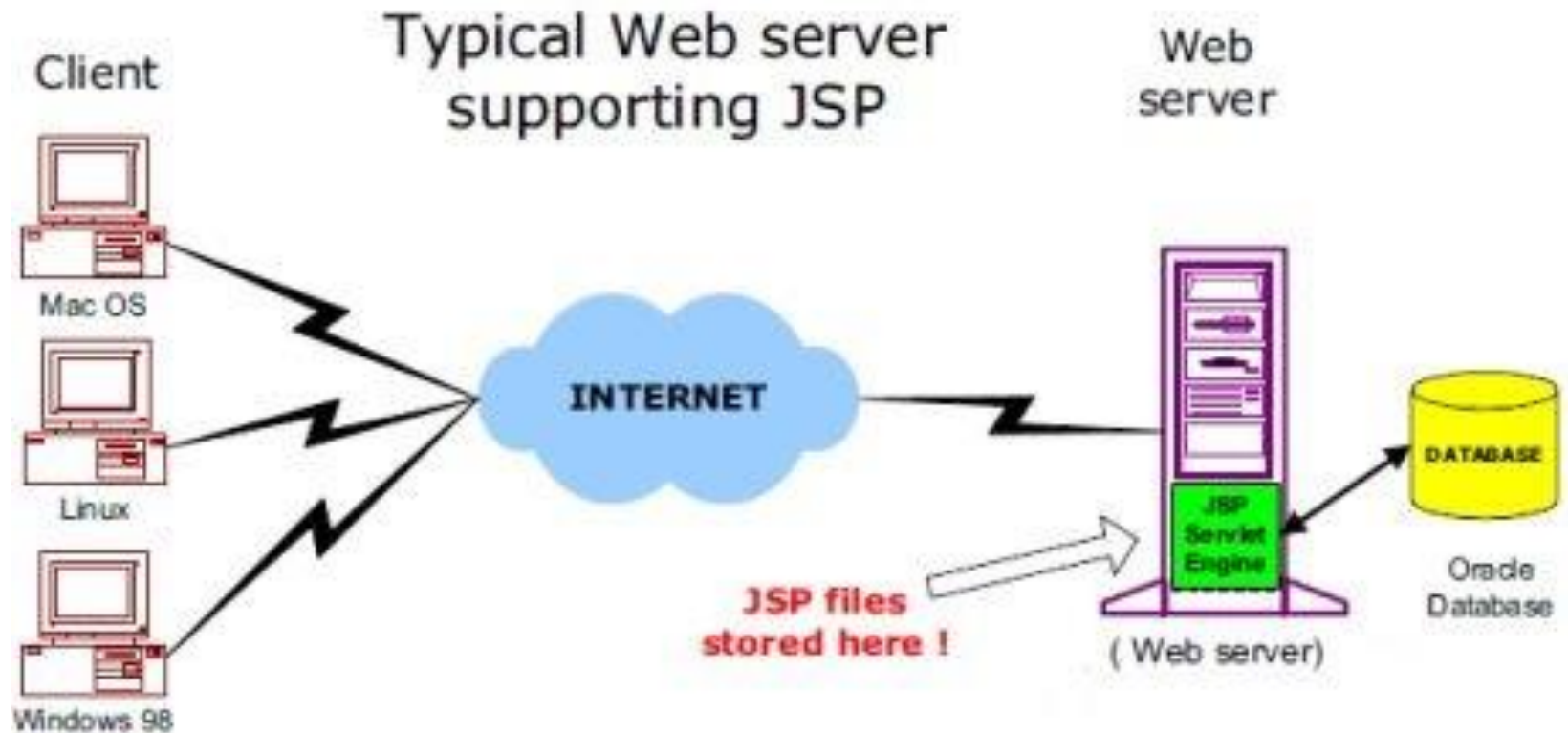
```
<h3 align="right" style="color:red"><jsp:include page="date.jsp" flush="true" /></h3>
```

```
</center>
```

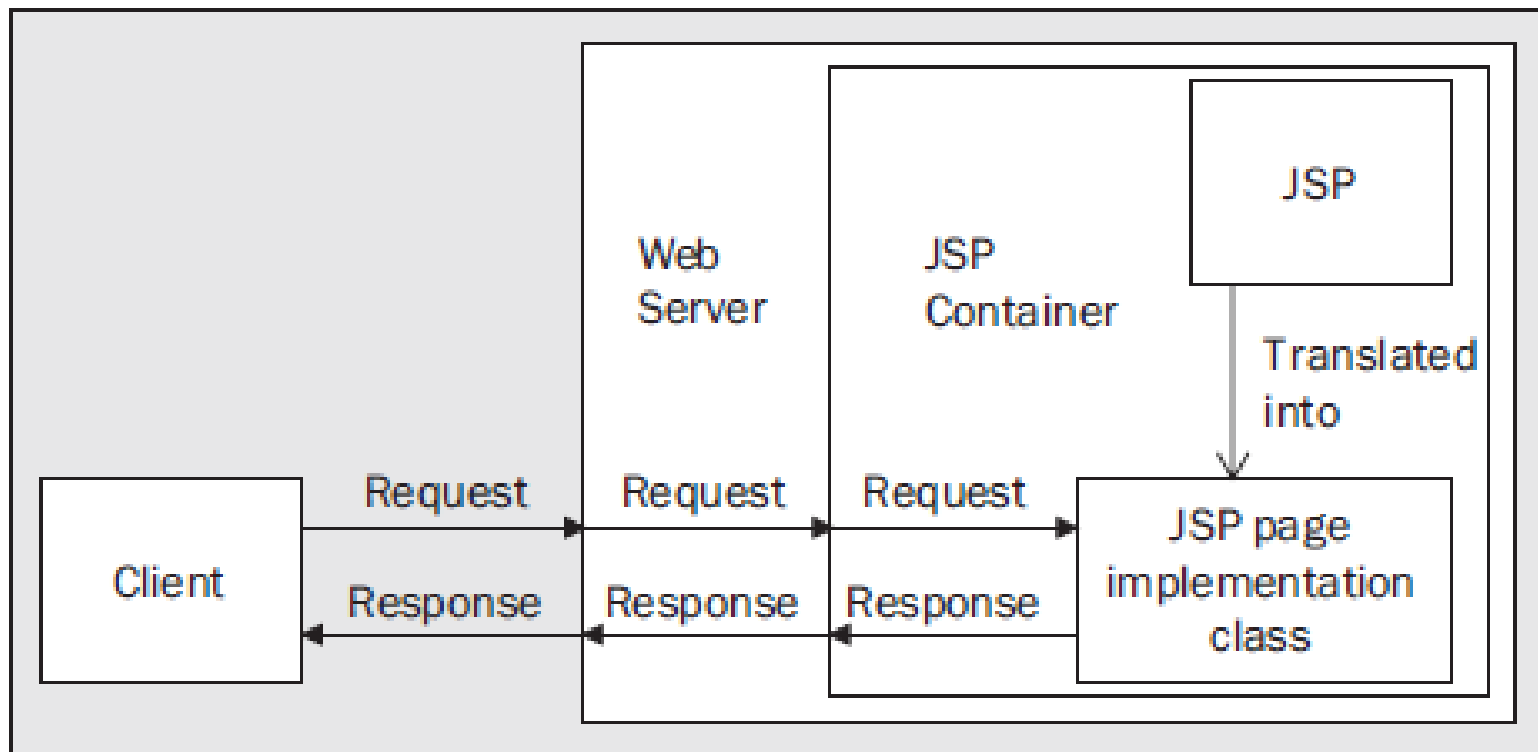
```
</body>
```

```
</html>
```

# JSP Container



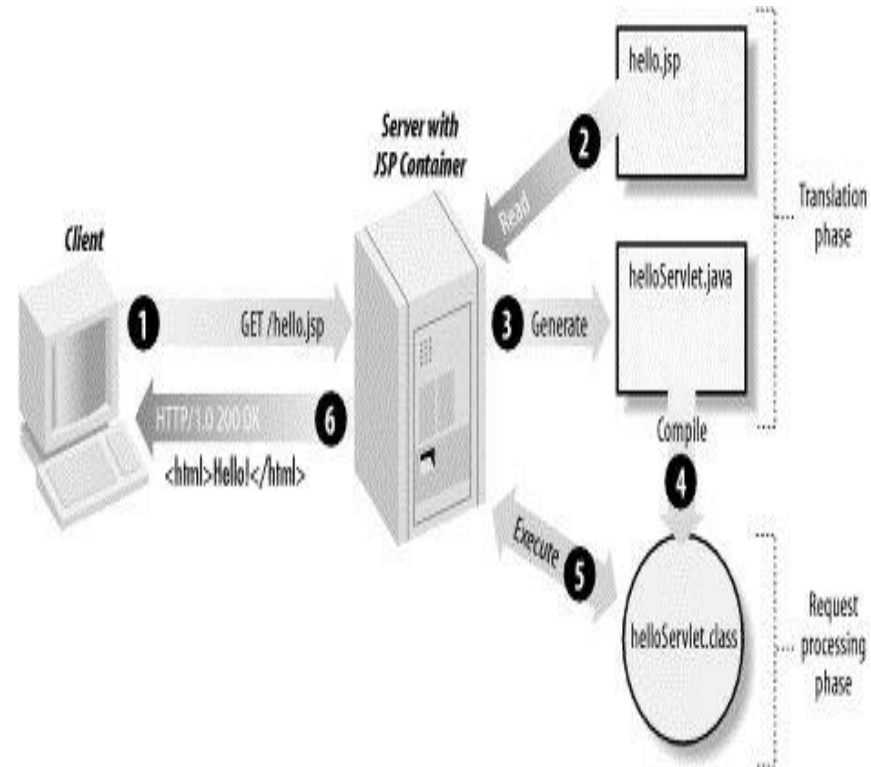
# JSP Container contd..



# JSP Processing

1. Browser sends an HTTP request to the web server.
2. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
3. The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to `println( )` statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
4. The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
5. A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
6. The web server forwards the HTTP response to your browser in terms of static HTML content.

Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.



# Implicit Objects

1. out

2.request

3.response

4.config

5.application

6.session

7.pageContext

8.page

9.exception



# Implicit Objects

Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables.

Object	Description
<b>request</b>	This is the <b>HttpServletRequest</b> object associated with the request
<b>response</b>	This is the <b>HttpServletResponse</b> object associated with the response to the client.
<b>out</b>	This is the <b>PrintWriter</b> object used to send output to the client.
<b>session</b>	This is the <b>HttpSession</b> object associated with the request.
<b>application</b>	This is the <b>ServletContext</b> object associated with application context.
<b>config</b>	This is the <b>ServletConfig</b> object associated with the page
<b>page</b>	This is simply a synonym for <b>this</b> , and is used to call the methods defined by the translated <b>servlet</b> class.
<b>exception</b>	The exception object allows the exception data to be accessed by designated JSP.

# request Object

- Each time a client requests a page the JSP engine creates a new object to represent that request.
- The request object has request scope. That means that the implicit request object is in scope until the response to the client is complete.
- It is an instance of `javax.servlet.HttpServletRequest` class.

# Methods

- **String getParameter(String name);**-This method is used to get the value of a request's parameter.
- **String[] getParameterValues(String name);**-It returns the array of parameter values.
- **String getPathInfo();**
- **Cookie[] getCookies();**- It returns an array of cookie objects received from the client.
- **URL getRequestURI();**-This method returns the URL of current JSP page.
- **String getMethod();** - It returns HTTP request method.

# response Object

It encapsulates the response to the web application client. It can set headers, set cookies for the client, and send a redirect response to the client.

It is an instance of `javax.servlet.http.HttpServletResponse` class and it has page scope.

# Methods

- `public void addHeader(String name, String value)`
- `public void addCookie(Cookie cookie)`
- `public void sendRedirect(String location)`: The **`sendRedirect()`** method can be used to redirect response to another resource, it may be servlet, jsp or html file.

# out Object

- The out implicit object is a reference to an output stream that you can use within scriptlets.
- The out object is an instance of `javax.jsp.JspWriter`. It has page scope.

# Methods

**void print(dataType dt)**

**void println(dataType dt)**

**void flush()**

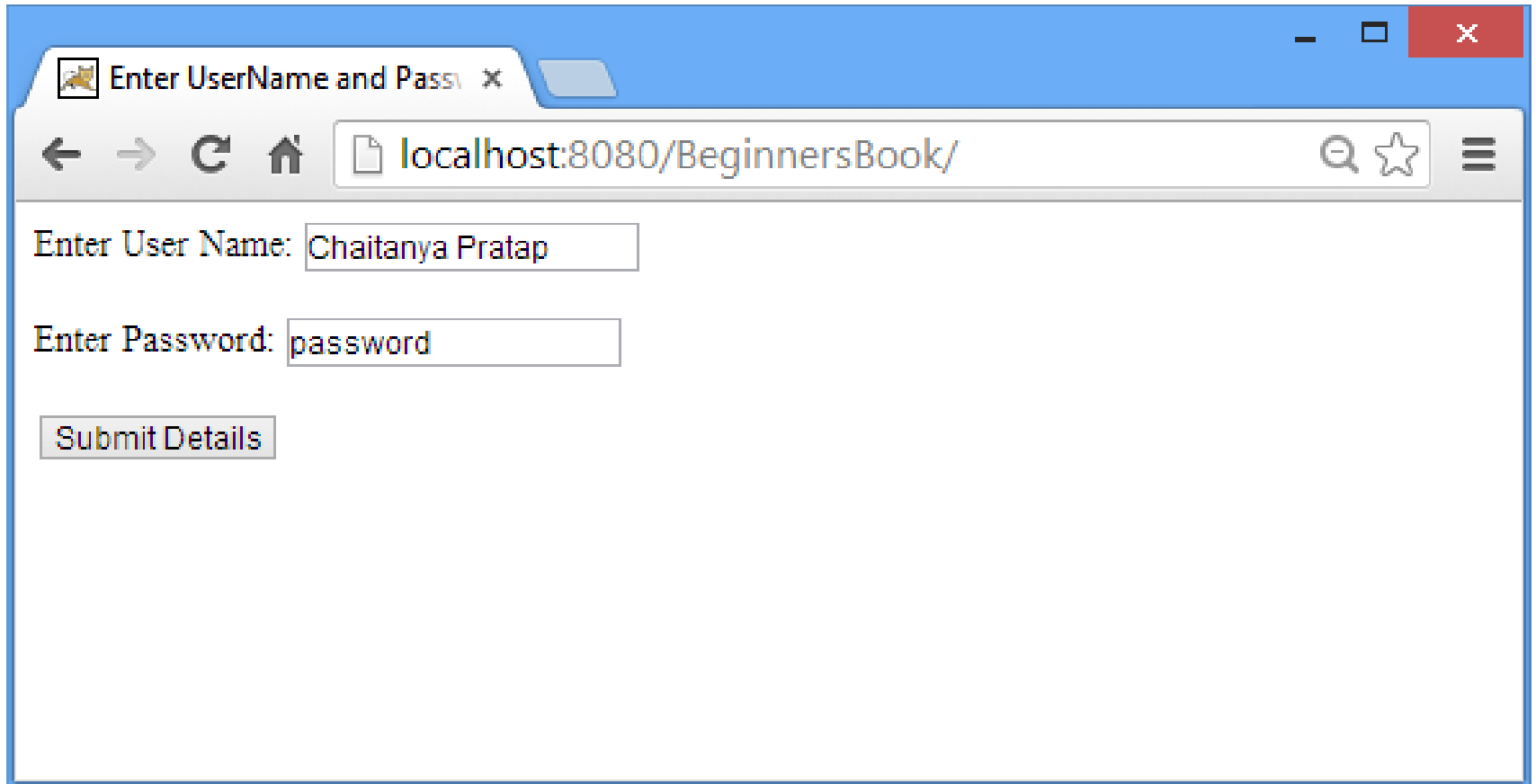
# session Object

HTTP is a stateless protocol. Using the session object, the page can store information about the client or the client's interaction. Information is stored in the session.

- `Object setAttribute(String name, Object value);`
- `Object getAttribute(String name);`
- `void removeAttribute(String name);`
- session' scope means, the JSP object is accessible from pages that belong to the same session from where it was created. The JSP object that is created using the session scope is bound to the session object. Implicit object session has the 'session' scope.



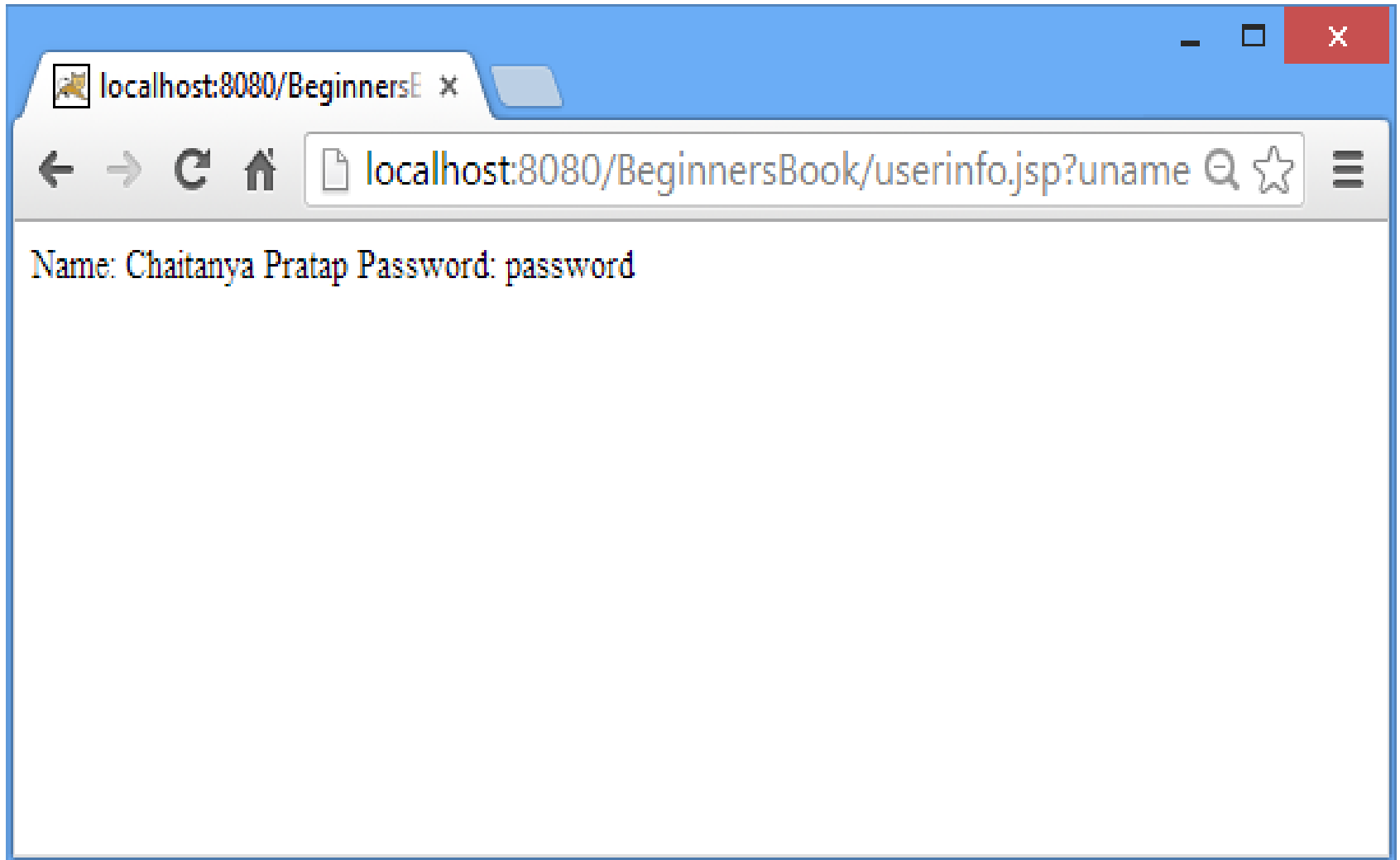
# PROBLEM 1



A screenshot of a web browser window. The title bar is blue and contains the text "Enter UserName and Pass" followed by a close button. The address bar shows "localhost:8080/BeginnersBook/" with search, star, and menu icons. The main content area has a white background and contains a login form with two text input fields and a submit button.

Enter User Name:

Enter Password:



# Example

index.html

```
<html>
```

```
<head>
```

```
<title>Enter UserName and Password</title>
```

```
</head>
```

```
<body>
```

```
<form action="userinfo.jsp">
```

```
Enter User Name: <input type="text" name="uname" /> <br>
```

```
<br>Enter Password: <input type="text" name="pass" /> <br>
```

```
<br><input type="submit" value="Submit Details"/> </form>
```

```
</body>
```

```
</html>
```

userinfo.jsp

<html>

<body>

<%

String username=request.getParameter("uname");

String password=request.getParameter("pass");

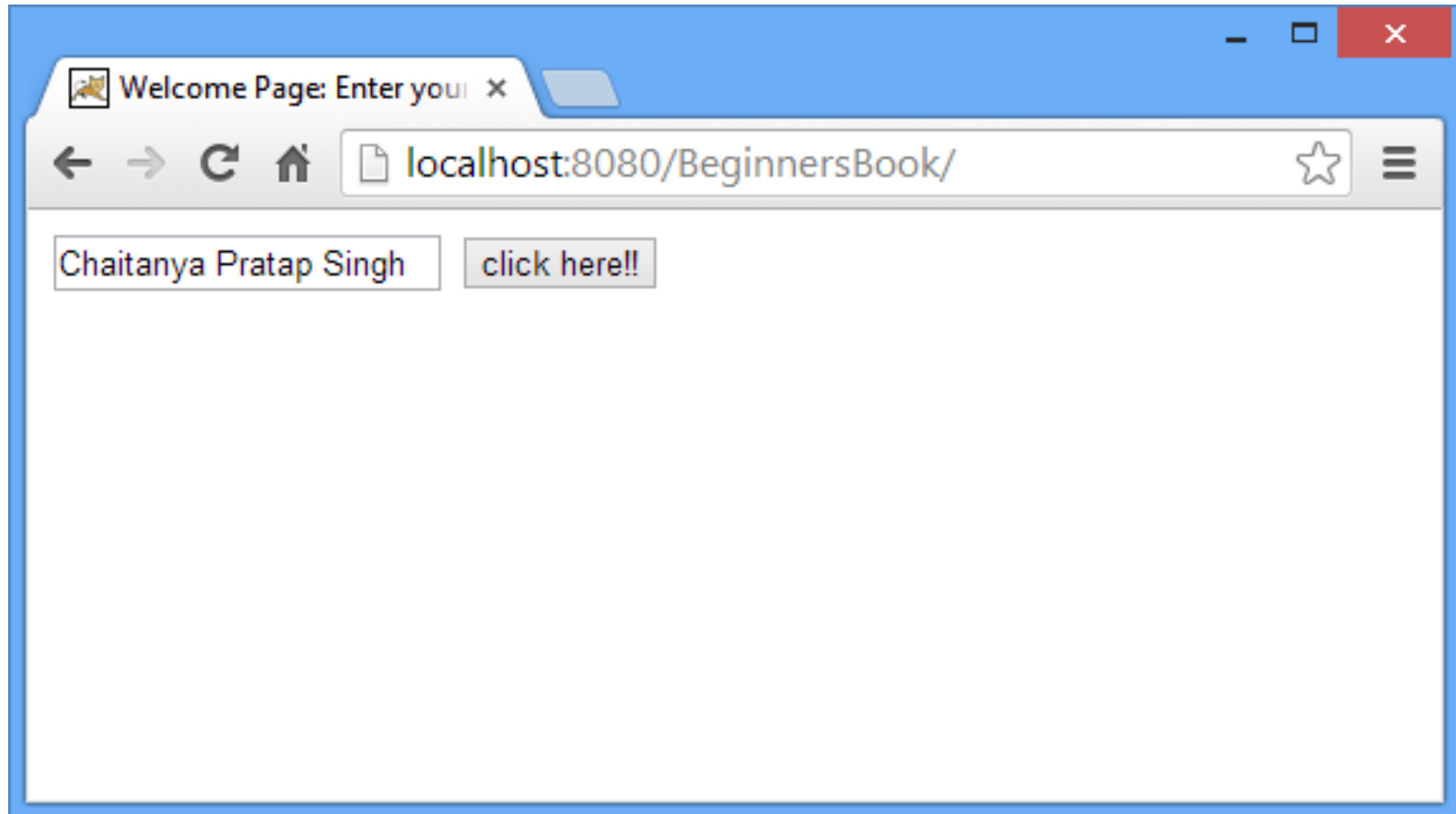
out.print("Name: "+username+" Password:  
"+password);

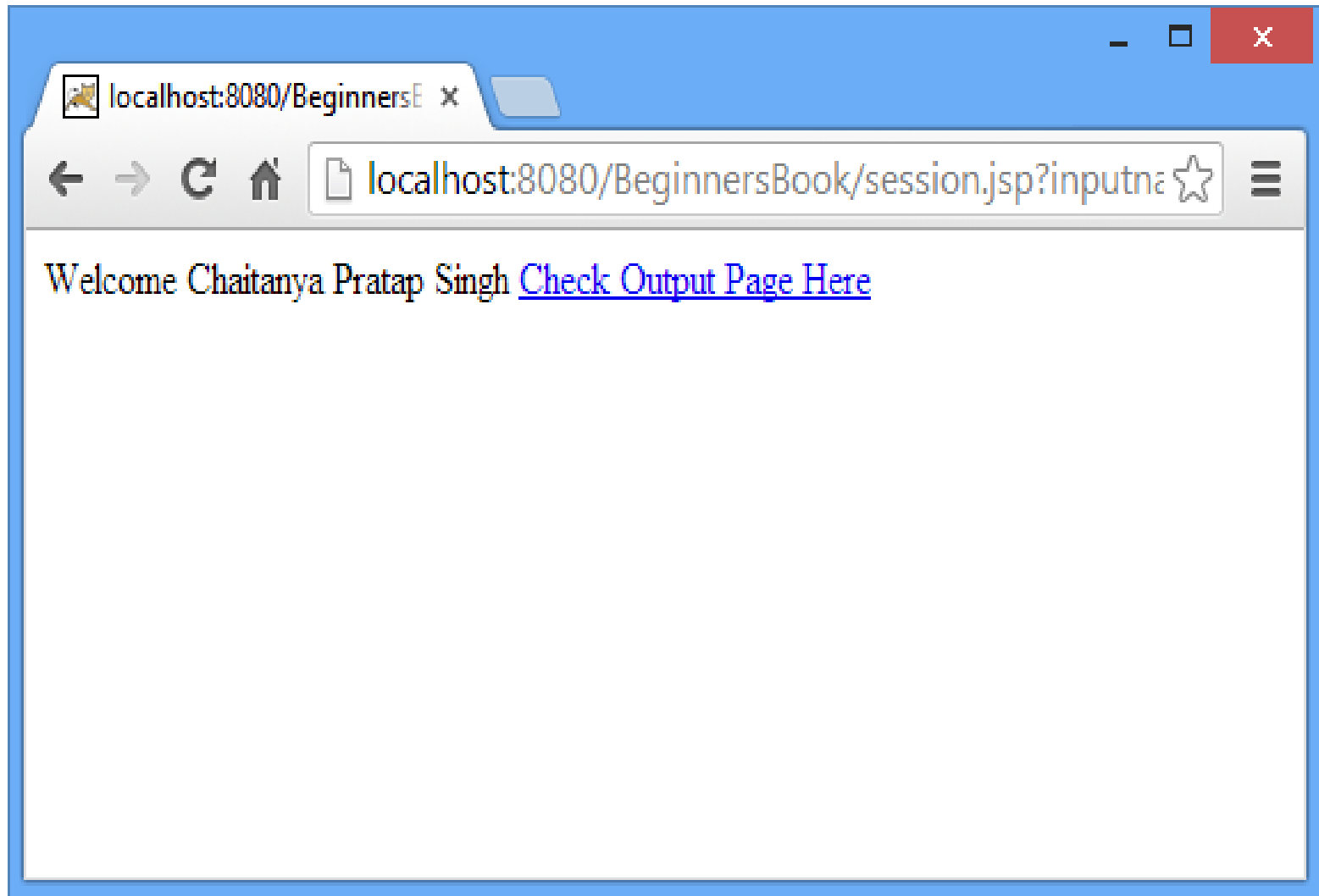
%>

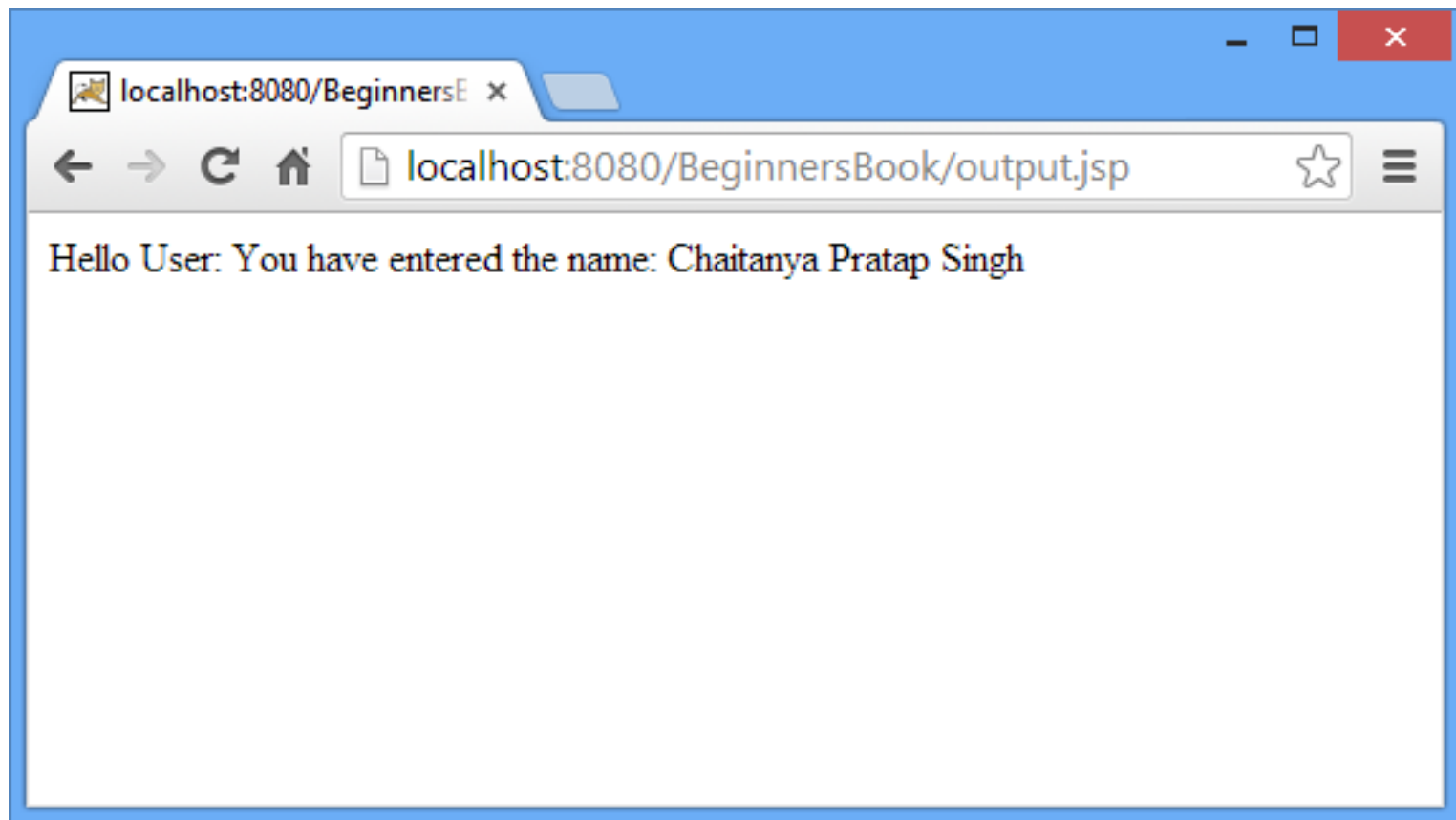
</body>

</html>

## PROBLEM 2







# Example

index.html

```
<html>
<head>
<title>Welcome Page: Enter your name</title>
</head>
<body>
<form action="session.jsp">
<input type="text" name="inputname">
  <input type="submit" value="click here!!">
</form>
</body>
</html>
```



session.jsp

```
<html>
<head>
<title>Passing the input value to a session variable</title>
</head>
<body>
<% String uname=request.getParameter("inputname");

out.print("Welcome "+ uname);
session.setAttribute("sessname",uname);
%>
<a href="output.jsp">Check Output Page Here </a>
</body>
</html>
```

output.jsp

```
<html>
  <head>
    <title>Output page: Fetching the value from session</title>
  </head>
  <body>
    <%
String name=(String)session.getAttribute("sessname");
  out.print("Hello User: You have entered the name: "+name);
%>
  </body>
</html>
```

index.html

<html>

<head>

<title>Login Page</title>

</head>

<body>

<form action="checkdetails.jsp">

UserId: <input type="text" name="id" /> <br>

<br>Password: <input type="text" name="pass" /> <br>

<br><input type="submit" value="Sign In!!"/> </form>

</body>

</html>

# checkdetails.jsp

This JSP page verifies the input id/pass against hard-coded values.

```
<html>
<head>
<title>Check Credentials</title>
</head>
<body>
<%
String uid=request.getParameter("id");
String password=request.getParameter("pass");
session.setAttribute("session-uid", uid);
if(uid.equals("Chaitanya") && password.equals("amity"))
{
response.sendRedirect("success.jsp");
}
else{ response.sendRedirect("failed.jsp");
}
%>
</body>
</html>
```

# success.jsp

This JSP page would execute if id/pass are matched to the hardcoded userid/password.

```
<html>
<head>
<title>Success Page</title>
</head>
<body>
<%
String data=(String)session.getAttribute("session-uid");
out.println("Welcome "+ data+"!!");
%>
</body>
</html>
```

# failed.jsp

The control will be redirected to this page if the credentials entered by user are wrong.

```
<html>
```

```
<head>
```

```
<title>Sign-in Failed Page</title>
```

```
</head>
```

```
<body>
```

```
<%
```

```
String data2=(String)session.getAttribute("session-uid");
```

```
out.println("Hi "+ data2+" . Id/Password are wrong. Please try  
Again.");
```

```
%>
```

```
</body>
```

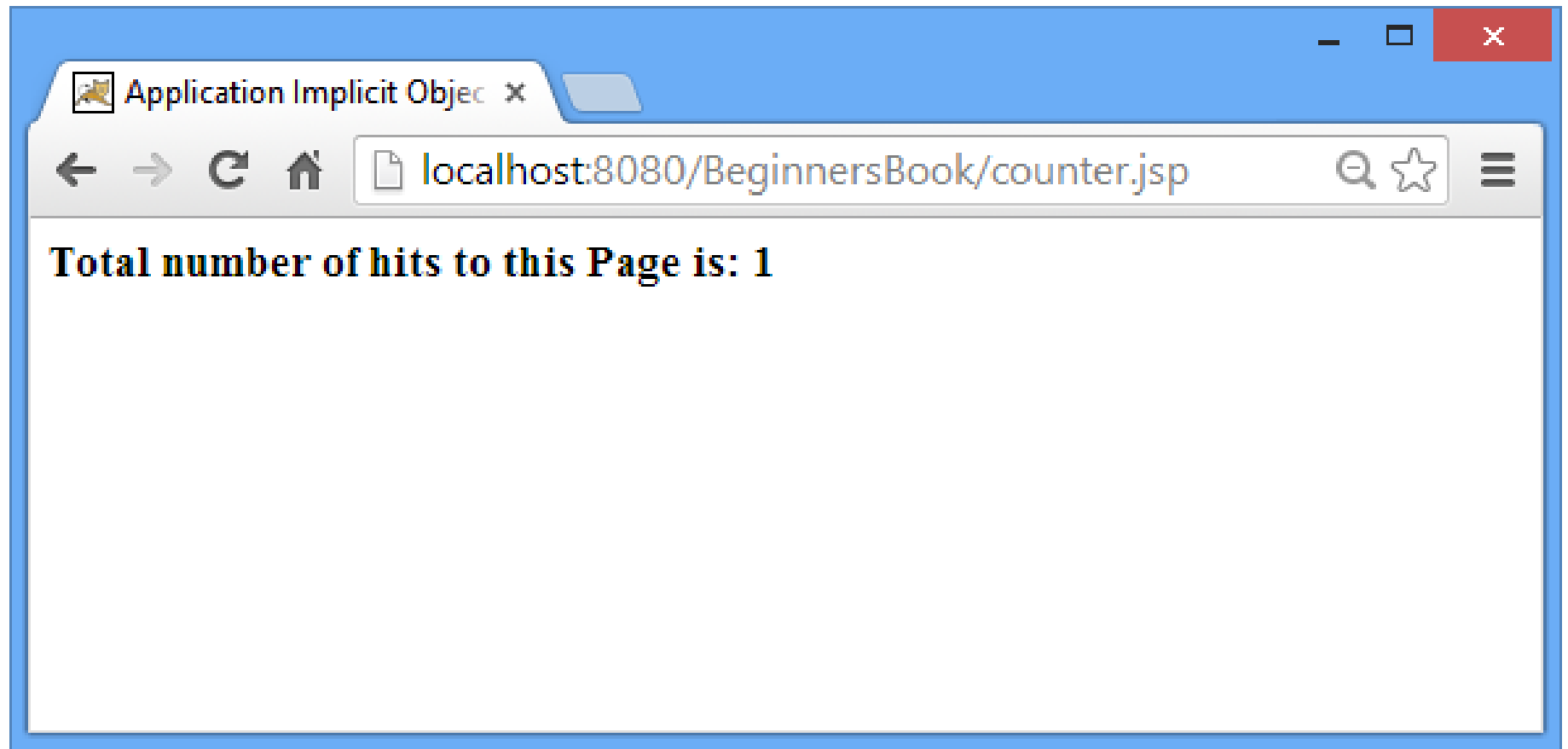
```
</html>
```

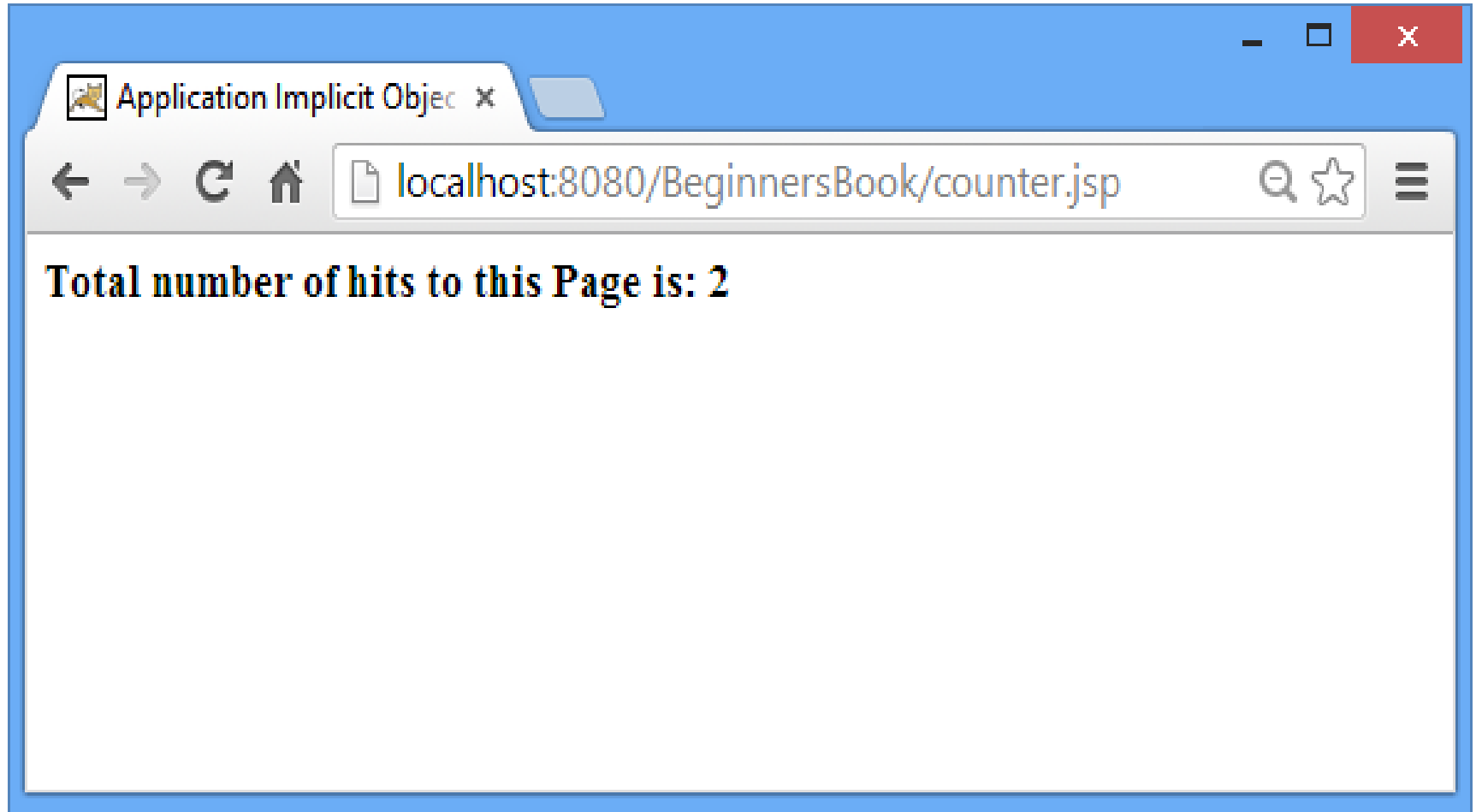
# application object

- In JSP, application is an implicit object of type `ServletContext`. This is an instance of `javax.servlet.ServletContext`. It is generated onetime by the web container when web application or web project is deployed on server.
- This object is used to acquire the initialization parameter from the configuration file (`web.xml`). It is used for allotting the attributes and the values of the attributes over all JSP page. This indicates that any attribute which is set by application object will be accessible to all the JSP pages. In the application scope, it is also used to get, set, and remove attribute.
- By adding an attribute to application, you can ensure that all JSP files that make up your web application have access to it.

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Application Implicit Object Example</title>
</head>
<body>
<% //Comment: This would return null for the first time
Integer counter= (Integer)application.getAttribute("numberOfVisits");
if( counter ==null || counter == 0 )
{ //Comment: For the very first Visitor
counter = 1;
}else
{ //Comment: For Others
counter = counter+ 1;
}
application.setAttribute("numberOfVisits", counter);
%>
<h3>Total number of hits to this Page is: <%= counter%>
</h3>
</body>
</html>
```







# GET method

- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows:
- `http://www.test.com/hello?key1=value1&key2=value2`
- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server.
- The GET method has size limitation: only 1024 characters can be in a request string.

# POST method

- A generally more reliable method of passing information to a backend program is the POST method.
- This method packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.

# x.html

```
<html>
```

```
<body>
```

```
<form action="main.jsp" method="GET">
```

```
First Name: <input type="text" name="first_name"><br />
```

```
Last Name: <input type="text" name="last_name" />
```

```
<input type="submit" value="Submit" />
```

```
</form>
```

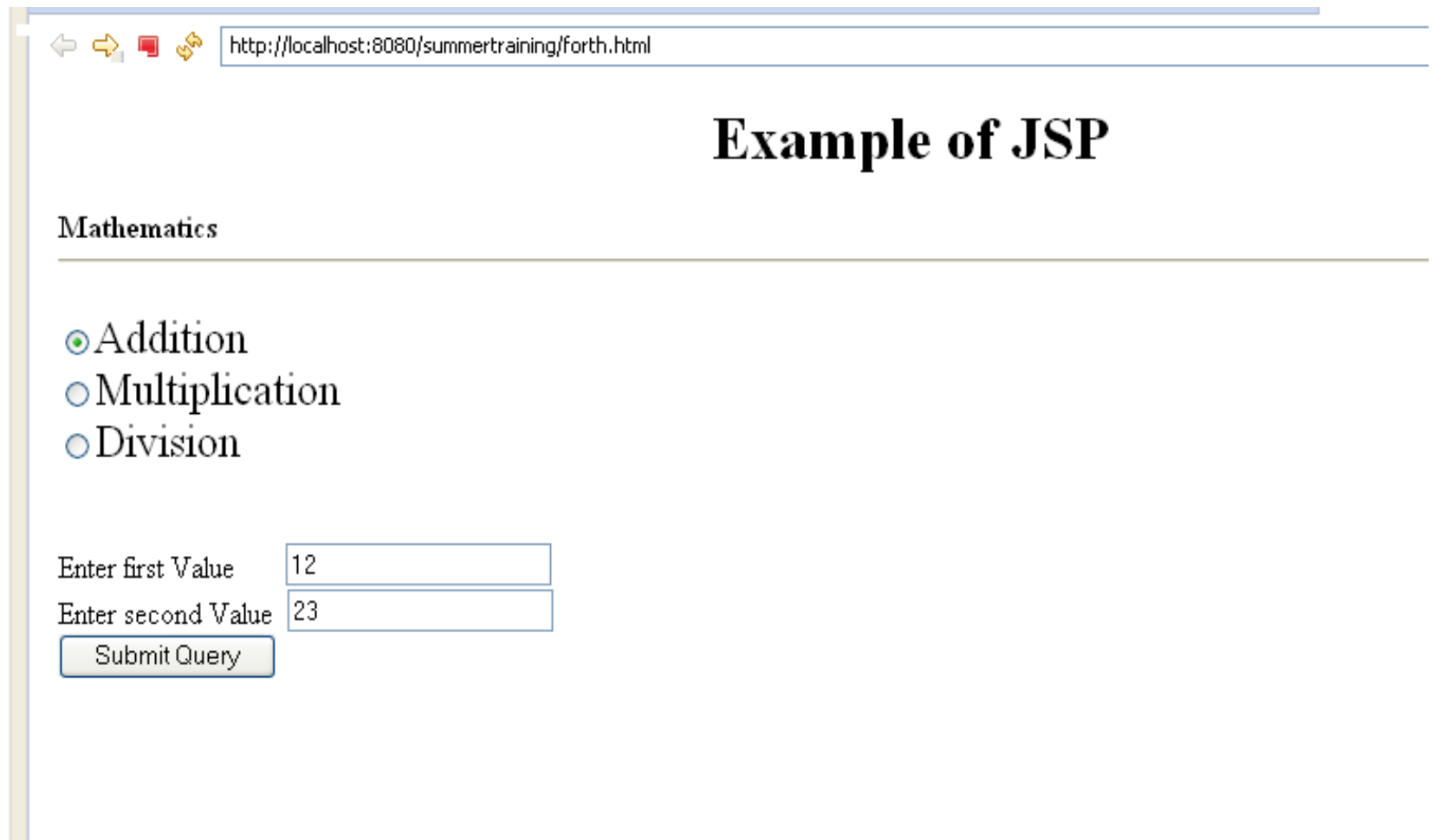
```
</body>
```

```
</html>
```

# main.jsp

```
<html>
<head>
<title>Using GET Method to Read Form Data</title>
</head>
<body>
<center><h1>Using GET Method to Read Form Data</h1>
<p><b>First Name:</b>  <%=
    request.getParameter("first_name")%></p>
<p><b>Last Name:</b>  <%=
    request.getParameter("last_name")%></p>
</body>
</html>
```

# PROBLEM 4



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/summertraining/forth.html`. The page content includes a title **Example of JSP** and a section header **Mathematics**. Below this, there are three radio buttons for selecting an operation: **Addition** (selected), **Multiplication**, and **Division**. Further down, there are two input fields: "Enter first Value" containing the number 12, and "Enter second Value" containing the number 23. A "Submit Query" button is located below the input fields.



http://localhost:8080/summertraining/a.jsp

## Result for add

Result is 35



```

<html>
<title>Sample Example </title>
<body>
<h1> <center> Example of JSP </center> </h1>
<b> Mathematics</b>
<hr>
<form method="post" action="a.jsp">
<font size=5 face="Times New Roman">
<input type="radio" name="a1" value="add" checked>Addition</input>
<br><input type="radio" name="a1" value="mul" >Multiplication</input>
<br><input type="radio" name="a1" value="div" >Division</input>
<br></font>
<br><br>Enter first Value &nbsp; &nbsp; &nbsp;
<input type="text" name="t1" value="">
<br>Enter second Value &nbsp;<input type="text" name="t2" value=""><br>
<input type="submit" name="result">
</form></body></html>

```

# a.jsp

```
<% @ page language="java"%>
<% @ page import="java.lang.*"%>
<html>
<body>
<H1><center>Result for <%=request.getParameter("a1")%></center></H1>
<%
int i=Integer.parseInt(request.getParameter("t1"));
int j=Integer.parseInt(request.getParameter("t2"));
int k=0;
String str=request.getParameter("a1");

if(str.equals("add"))
    k=i+j;
if(str.equals("mul"))
    k=i*j;
if(str.equals("div"))
    k=i/j;
%>
Result is <%=k%>
</body>
</html>
```

# exception object (index.html)

```
<html>
<head>
<title>Enter two Integers for Division</title>
</head>
<body>
<form action="division.jsp">
Input First Integer:<input type="text" name="firstnum" />
Input Second Integer:<input type="text" name="secondnum" />
  <input type="submit" value="Get Results"/> </form>
</body>
</html>
```

# division.jsp

```
<%@ page errorPage="exception.jsp" %>
<%
    String num1=request.getParameter("firstnum");
    String num2=request.getParameter("secondnum");
    int v1= Integer.parseInt(num1);
    int v2= Integer.parseInt(num2);
    int res= v1/v2;
    out.print("Output is: "+ res);
%>
```

# exception.jsp

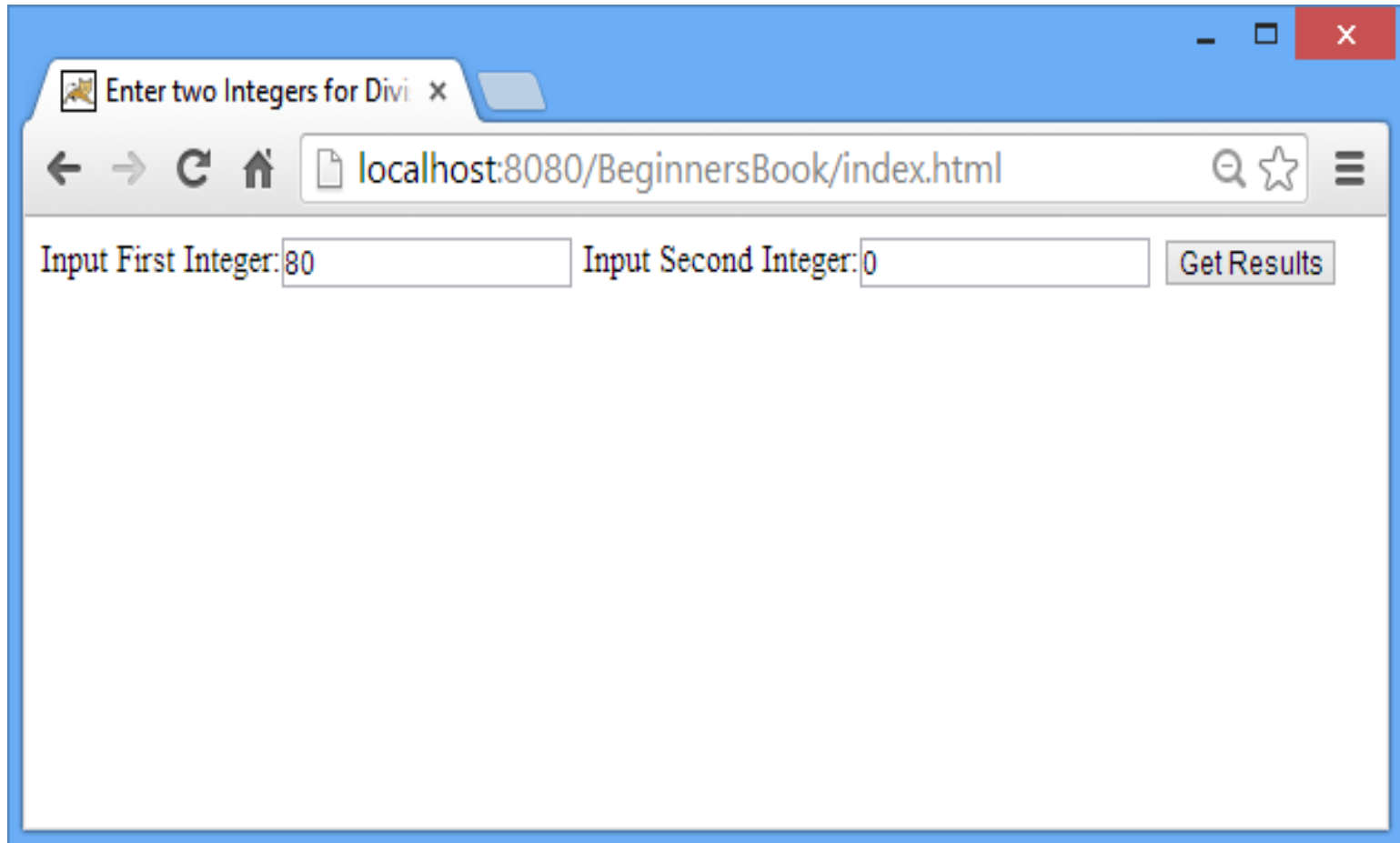
```
<%@ page isErrorPage="true" %>
```

Got this Exception:

```
<%= exception %>
```

Please correct the input data.

# Output



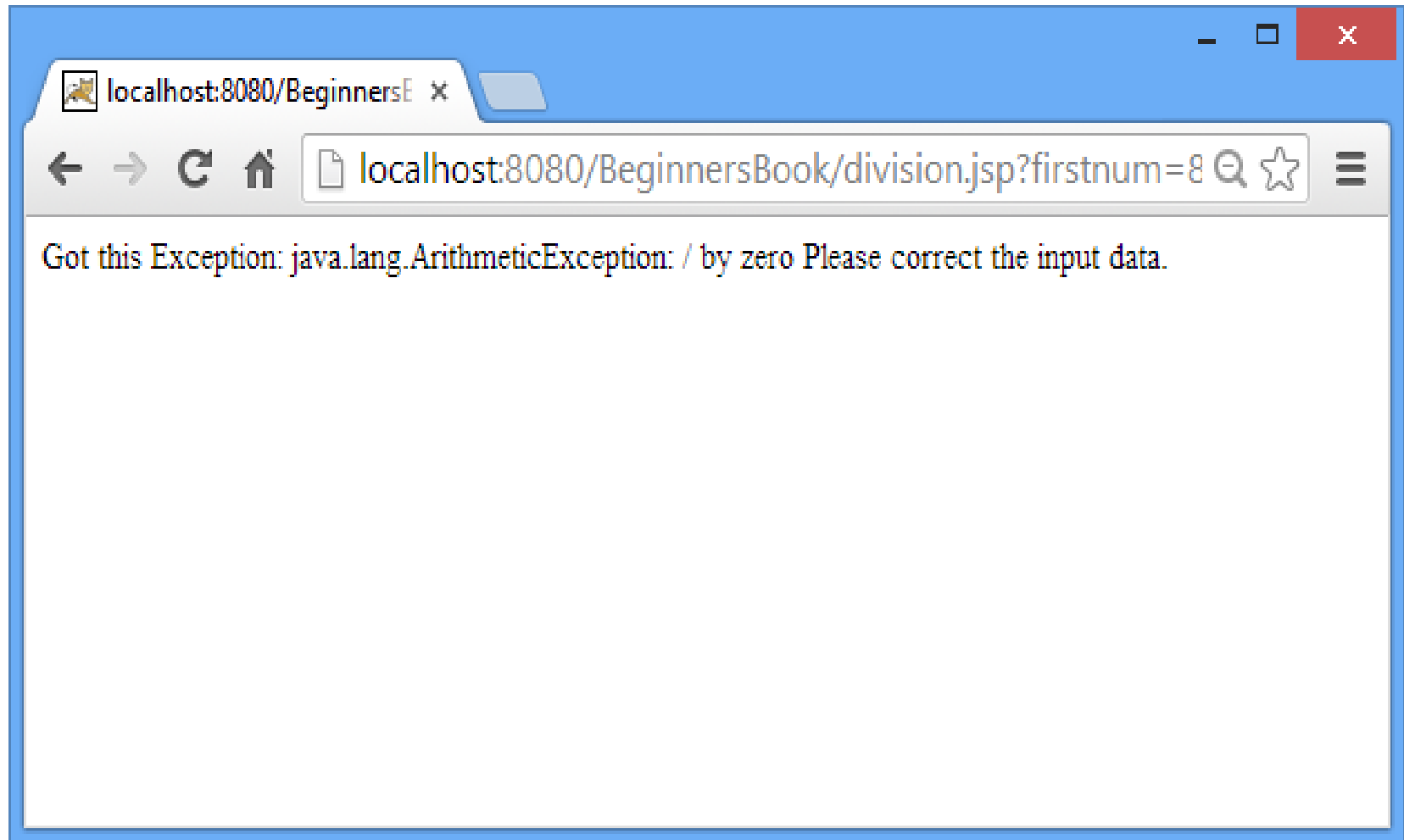
A screenshot of a web browser window. The title bar is blue and contains the text "Enter two Integers for Divi" followed by a close button. The address bar shows the URL "localhost:8080/BeginnersBook/index.html". The main content area has a light blue background and contains two input fields. The first field is labeled "Input First Integer:" and contains the value "80". The second field is labeled "Input Second Integer:" and contains the value "0". To the right of these fields is a button labeled "Get Results".

Enter two Integers for Divi x

localhost:8080/BeginnersBook/index.html

Input First Integer: 80 Input Second Integer: 0 Get Results

# Output



# pageContext Object

- This object is intended as a means to access information about the page while avoiding most of the implementation details.

Here we are simply asking user to enter login details.

```
<html>
<head>
<title> User Login Page – Enter details</title>
</head>
<body>
<form action="validation.jsp">
Enter User-Id: <input type="text" name="uid"><br>
Enter Password: <input type="text" name="upass"><br>
<input type="submit" value="Login">
</form>
</body>
</html>
```



# validation.jsp

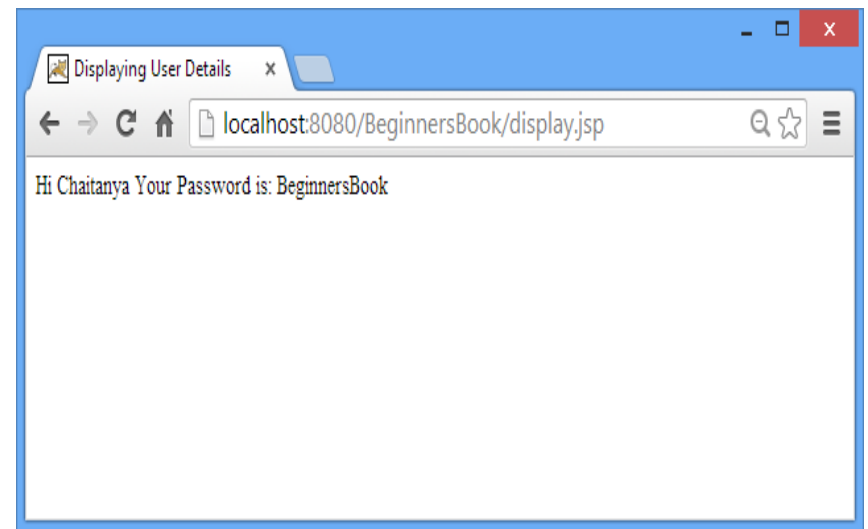
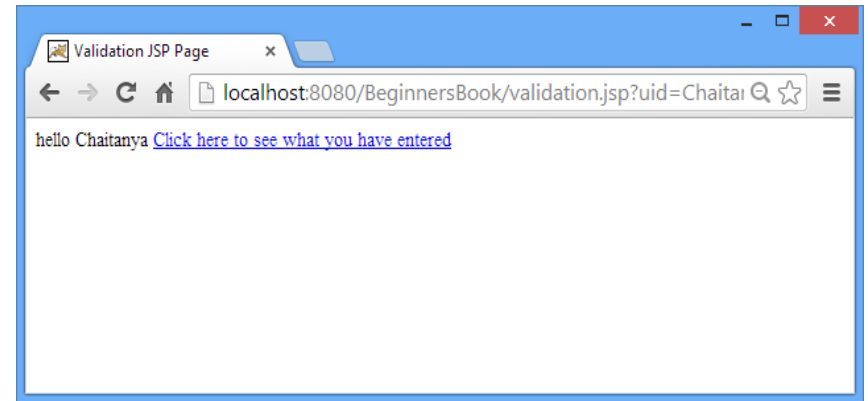
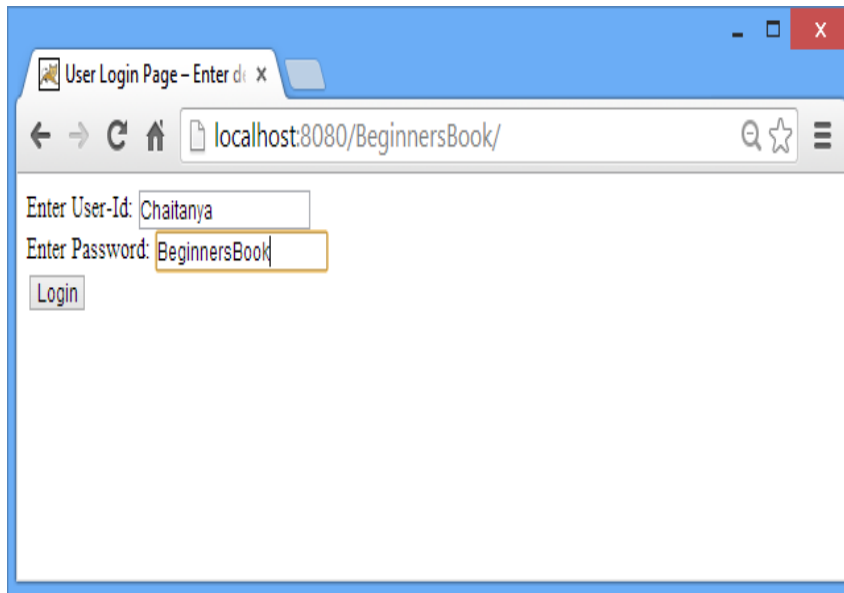
- In this page we are storing user's credentials using **pageContext** implicit object with the **session scope**, which means we will be able to access the details till the user's session is active. We can also store the attribute using other scope parameters such as page, application and request.

```
<html>
<head>
  <title> Validation JSP Page</title>
</head>
<body>
<%
String id=request.getParameter("uid");
String pass=request.getParameter("upass");
out.println("hello "+id);
pageContext.setAttribute("UName", id, pageContext.SESSION_SCOPE);
pageContext.setAttribute("UPassword", pass, pageContext.SESSION_SCOPE);
%>
<a href="display.jsp">Click here to see what you have entered </a>
</body>
</html>
```

# display.jsp

- In this JSP page we are fetching the stored attributes using `getAttribute` method. The point to note here is that we have stored the attributes with session scope so we must need to specify scope as session in order to fetch those attribute's value.

```
<html>
<head>
<title>Displaying User Details</title>
</head>
<body>
<%
String username= (String) pageContext.getAttribute("UName",
    pageContext.SESSION_SCOPE);
String userpassword= (String) pageContext.getAttribute("UPassword",
    pageContext.SESSION_SCOPE);
out.println("Hi "+username);
out.println("Your Password is: "+userpassword);
%>
</body>
</html>
```



Directive	Attribute	Description
Page	import	Lists the Java packages to be imported into the page. Just as with a Java source file, the Java code embedded in a JSP page must import the packages of the classes used with the code. Multiple package statements are delimited by commas, for example <code>import="java.io.*,java.util.*"</code> .
	session	The valid values are "true" or "false". The default value is "true". If "true", the page participates in a session; if "false", then it does not, and cannot access any session information. Sessions are covered later in the chapter.
	isThreadSafe	Whether the page is thread-safe or not. If "true", the container can use the JSP for multiple concurrent request threads. The default is "true".
	info	An arbitrary string. This can have any value. It is provided so that the JSP can provide information to a management tool about its contents, purpose, name, etc.
	errorPage	The URL of the web page that should be sent to the client if an error occurs in a page.
	isErrorPage	Whether the current page is an error page. The default is false.

include	contentType	Defines the content type of the page. The content type can appear as a simple type specification, or as a type specification and a charset. The default value is "text/html" for JSP-style JSP tags and "text/xml" for XML-style JSP tags. When including the charset, the syntax for the attribute is <code>contentType="text/html; charset=char_set_identifier"</code> . Whitespace can follow the semicolon in the attribute value. Charsets indicate how written characters are encoded, so that pages can support languages that use different scripts. Information about charsets can be found at <a href="http://www.w3.org/TR/REC-html40/charset.html">http://www.w3.org/TR/REC-html40/charset.html</a> .
	pageEncoding	The charset of the current page. The default is ISO-8859-1 (Latin script) for JSP-style and UTF-8 (an 8-bit Unicode encoding) for XML-style tags.
	file	The file to be included at the current position in the file. The included file can be any HTML or JSP page or fragment of a page. The file is specified using a URI to a file within the web application.

# Comments

- HTML comments within the JSP and those comments will appear in the page received by the client browser.

`<!-- This comment will appear in the client's browser —>`

- JSP-specific comments that will not appear in the page output to the client

`<%-- This comment will NOT appear in the client's browser —%>`

# Action Element

Action elements are basically predefined functions and there are following JSP actions available:

## Syntax

## Purpose

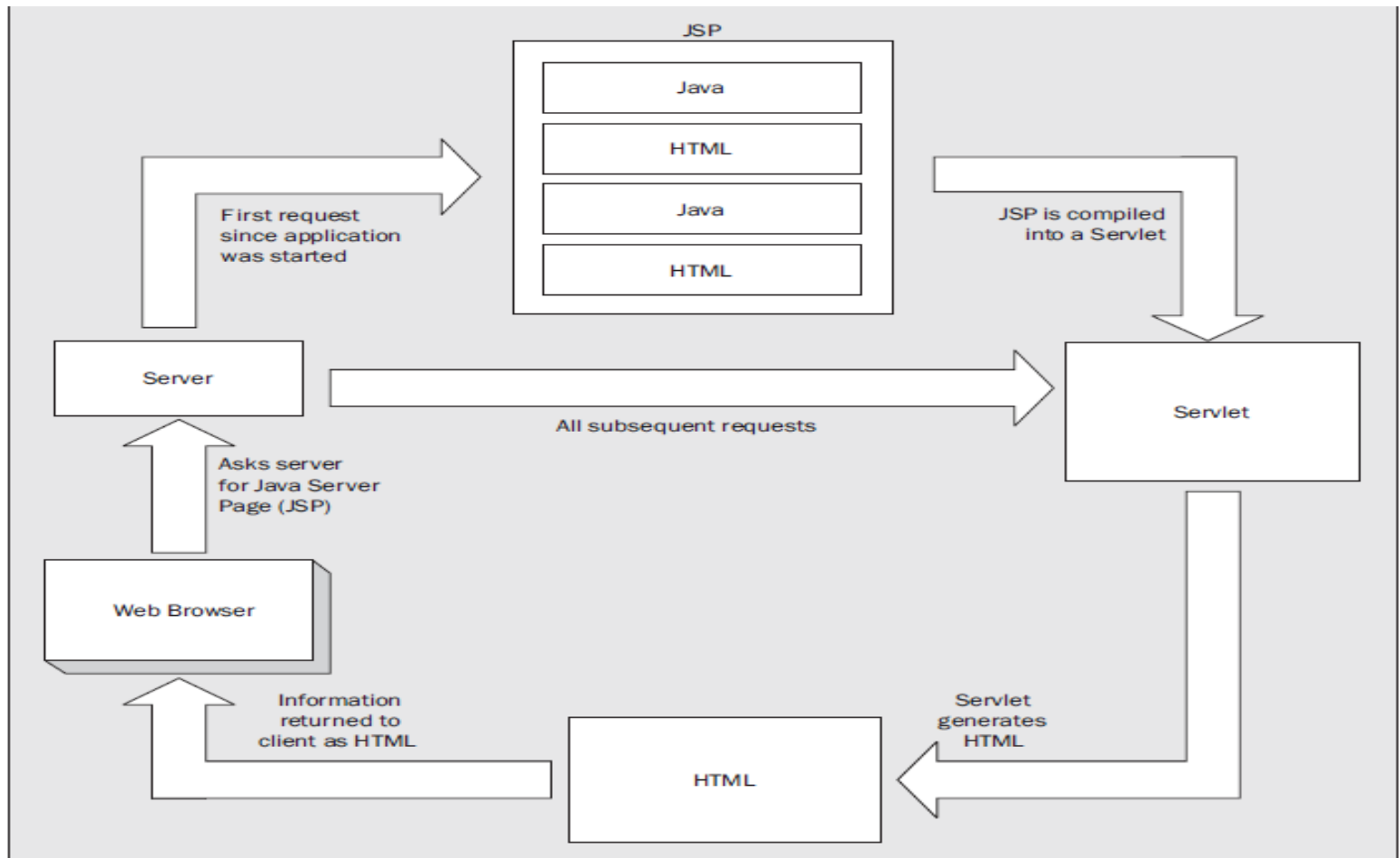
jsp:include		Includes a file at the time the page is requested
jsp:useBean		Finds or instantiates a JavaBean
jsp:setProperty		Sets the property of a JavaBean
jsp:getProperty		Inserts the property of a JavaBean into the output
jsp:forward		Forwards the requester to a new page
jsp:plugin	OBJECT or EMBED	Generates browser-specific code that makes an tag for the Java plugin
jsp:element		Defines XML elements dynamically.
jsp:attribute	attribute.	Defines dynamically defined XML element's
jsp:body		Defines dynamically defined XML element's body.
jsp:text	documents.	Use to write template text in JSP pages and

# Java Server Pages

- ☐ Creation—The developer creates a JSP source file that contains HTML and embedded Java code.
- ☐ Deployment—The JSP is installed into a server. This can be a full J2EE server or a stand-alone web server.
- ☐ Translation and compilation—The JSP container translates the HTML and Java code into a Java code source file. This file is then compiled into a Java class that is executed by the server. The class file created from the JSP is known as the JSP page implementation class.



# JSP Processing contd..



# Basic JSP Lifecycle

Once compilation(Parsing the JSP, Turning the JSP into a servlet, Compiling the servlet) is complete, the JSP lifecycle has these phases:

- ☐ Loading and instantiation—The server finds the Java class for the JSP page and loads it into the Virtual Machine. After the class is loaded, the JVM creates one or more instances of the page. This can occur right after loading, or it can occur when the first request is made.
- ☐ Initialization—The JSP page is initialized. If you need to execute code during initialization, you can add a method to the page that will be called during initialization.
- ☐ Request processing—The page responds to a request. After performing its processing, a response is returned to the client. The response consists solely of HTML tags or other data; none of the Java code is sent to the client.
- ☐ End of life—The server stops sending requests to the JSP. After all current requests are finished processing, any instances of the class are released. If you need code to execute and perform any cleanup actions, you can implement a method that will be called before the class instance is released.