CSC584 Enterprise Programming

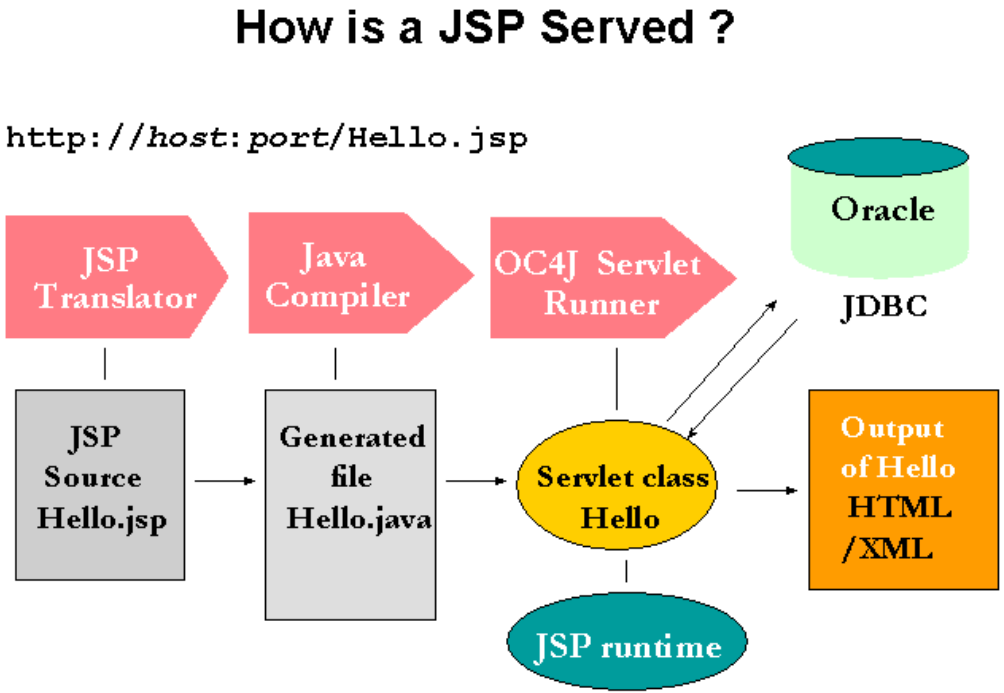
CHAPTER 4 – JSP

Chapter 4 Outline

- What is JSP?
- Advantages of JSP
- MVC –JSP?
- JSP Constructs
- JSP Predefined Variables
- JSP Directives
- JSTL Tags

What is JSP?

- Mostly HTML page, with extension .jsp
- Include JSP tags to enable dynamic content creation.
- Translation: JSP → Servlet class
- JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP.



What is a JSP?

```
<html>
  <body>
    <jsp:useBean.../>
    <jsp:getProperty.../>
    <jsp:getProperty.../>
  </body>
</html>
```

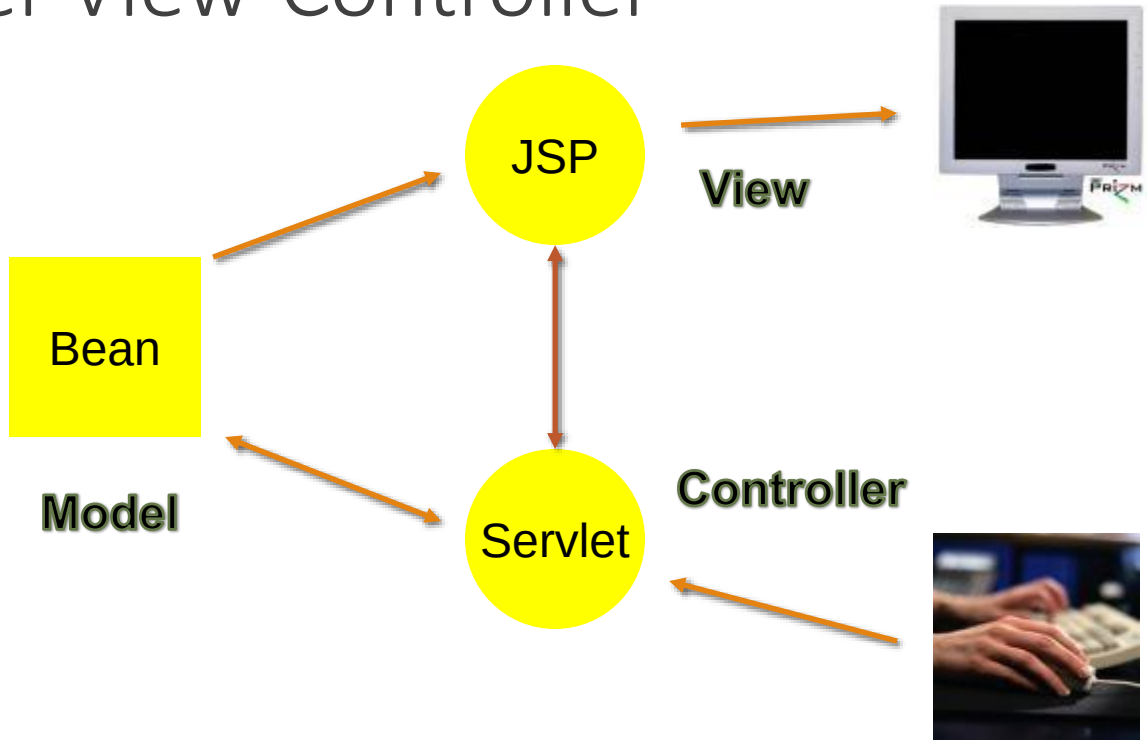
Advantages

- Code -- Computation
- HTML -- Presentation
- Separation of Roles (easy to maintain)
  - JSP can be easily managed because we can easily separate our business logic with presentation logic.
- Fast Development: No need to recompile and redeploy
  - If JSP page is modified, we don't need to recompile and redeploy the project.

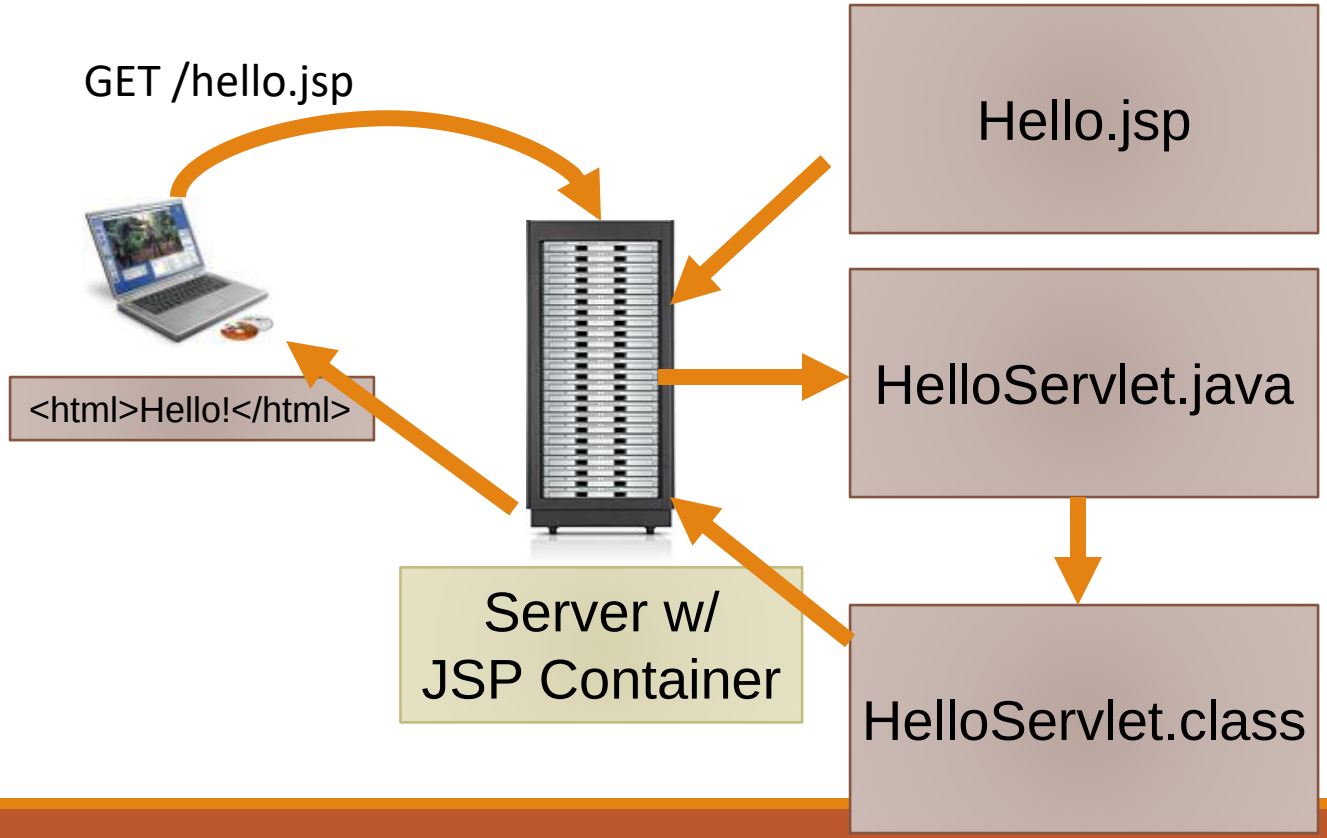
Model-View-Controller

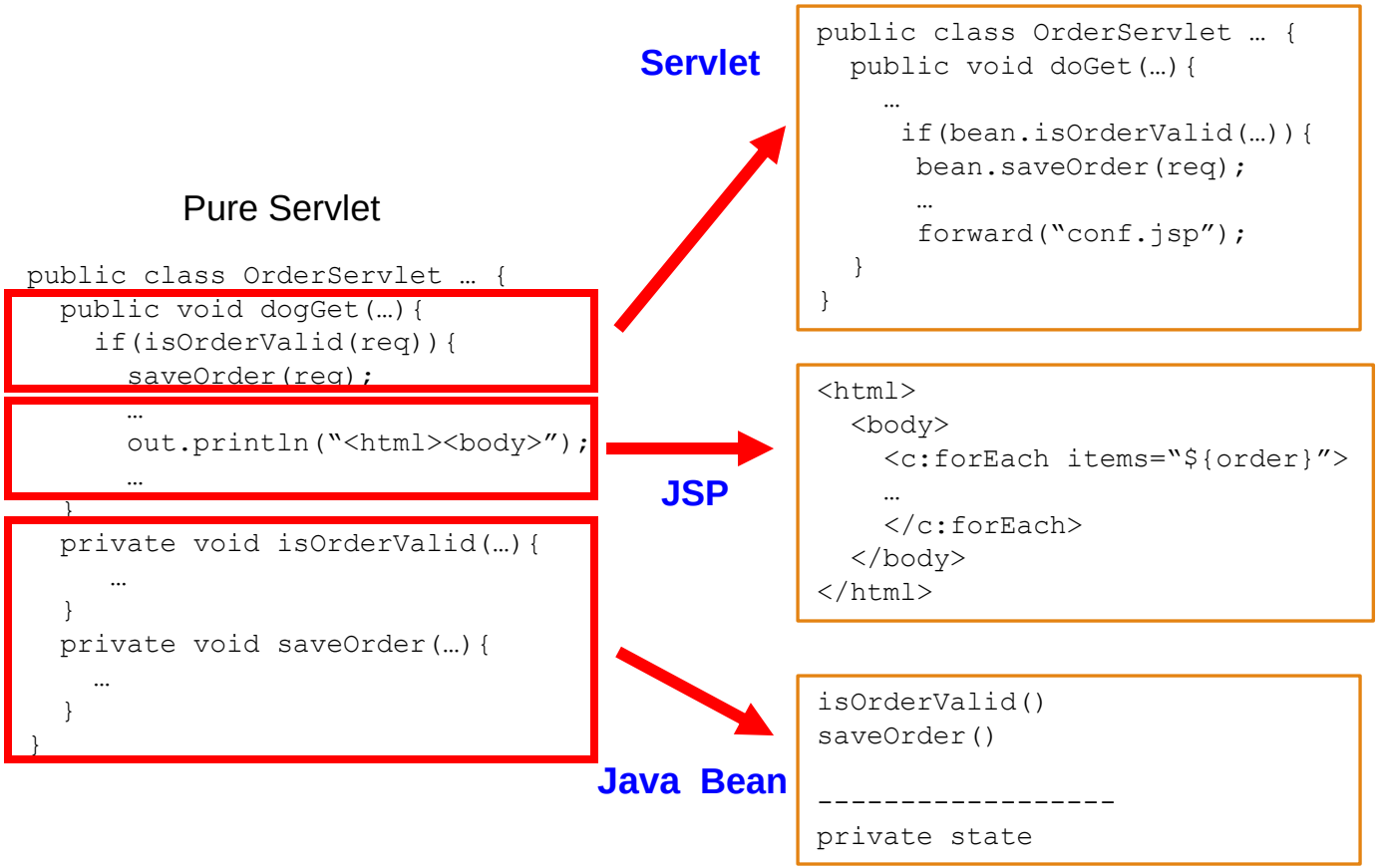
- A Design Pattern
  - Controller -- receives user interface input, updates data model
  - Model -- represents state of the world (e.g. shopping cart)
  - View -- looks at model and generates an appropriate user interface to present the data and allow for further input

Model-View-Controller



JSP Big Picture




Pure Servlet
public class OrderServlet … {
public void dogGet(…){
if(isOrderValid(req)){
saveOrder(req);
…
out.println("<html><body>");
…
}
private void isOrderValid(…){
…
}
private void saveOrder(…){
…
}
}
Servlet
public class OrderServlet … {
public void doGet(…){
…
if(bean.isOrderValid(…)){
bean.saveOrder(req);
…
forward("conf.jsp");
}
}
JSP
<html>
<body>
<c:forEach items="${order}">
…
</c:forEach>
</body>
</html>
Java Bean
isOrderValid()
saveOrder()
----
private state

A JSP File

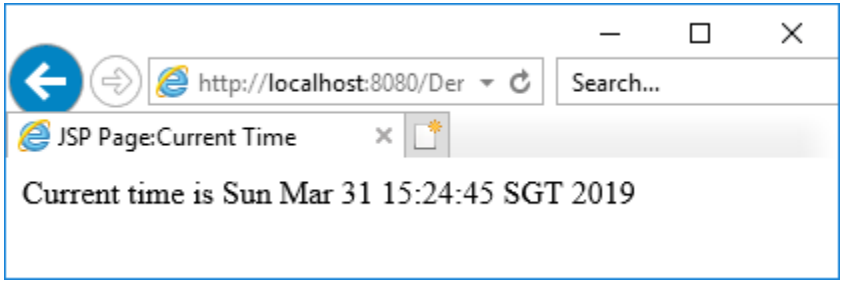

<%@ page language="java" contentType="text/html" %>
JSP element
<html>
<body bgcolor="white">
template text
<jsp:useBean
id="userInfo"
class="com.ora.jsp.beans.userinfo.UserInfoBean">
<jsp:setProperty name="userInfo" property="*"/>
</jsp:useBean>
JSP element
The following information was saved:

User Name:
template text
<jsp:getProperty name="userInfo"
property="userName"/>
JSP element
Email Address:
template text
<jsp:getProperty name="userInfo"
property="emailAddr"/>
JSP element

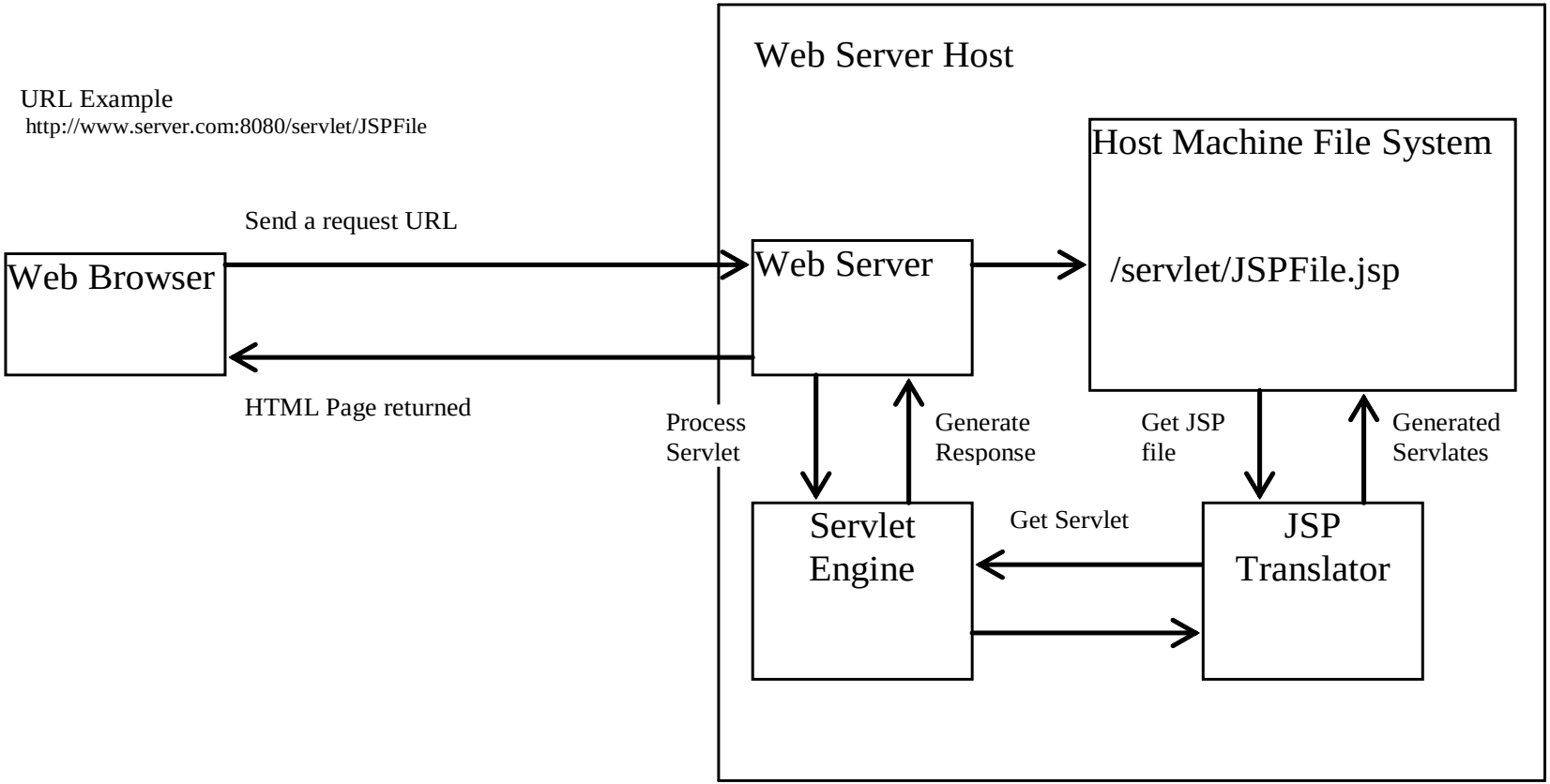
</body>
</html>
template text

A Simple JSP

```
<!--currentTime.jsp-->
<html>
   <head>
      <title>JSP Page:Current Time</title>
   </head>
   <body>
      Current time is <%= new java.util.Date() %>
   </body>
</html>
```



How Is a JSP Processed?



JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP expression is used to insert a Java expression directly into the output. It has the following form:

<%= Java-expression %>

The expression is evaluated, converted into a string, and sent to the output stream of the servlet.

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP scriptlet enables you to insert a Java statement into the servlet's jspService method, which is invoked by the service method. A JSP scriptlet has the following form:

<%= Java statement %>

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the resultant servlet. They are *expressions*, *scriptlets*, and *declarations*.

expression

scriptlet

declaration

A JSP declaration is for declaring methods or fields into the servlet. It has the following form:

<%! Java method or field declaration %>

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

JSP Comment

HTML comments have the following form:

<!-- HTML Comment -->

If you don’t want the comment appear in the resultant HTML file, use the following comment in JSP:

<%-- JSP Comment --%>

Example 4.1 Computing Factorials

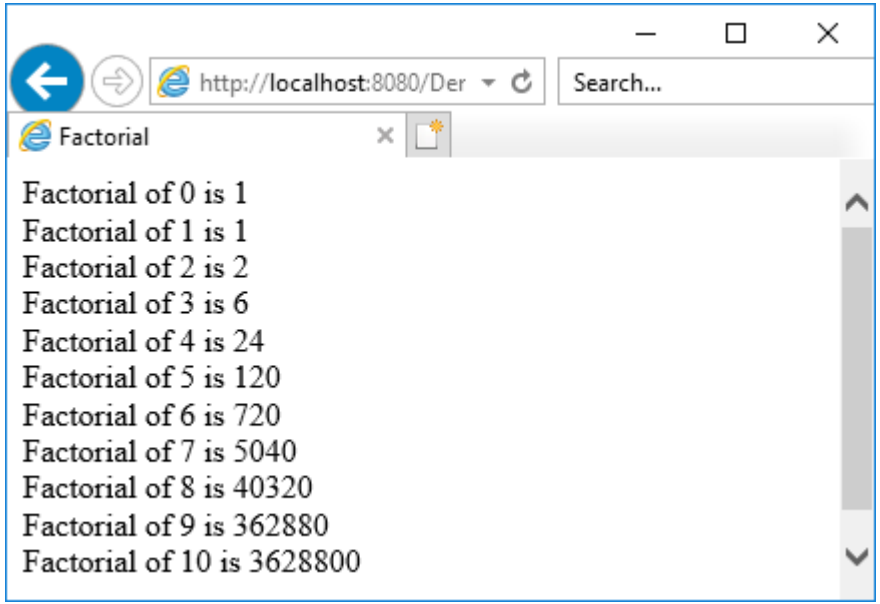
```
<html>
    <head>
        <title>Factorial</title>
    </head>
    <body>
      <%  for (int i = 0; i <= 10; i++) { %>
        Factorial of <%= i %> is
      <%= computeFactorial(i) %> <br />
      <%  } %>

    <%! private long computeFactorial(int n) {
          if (n == 0)
            return 1;
          else
            return n * computeFactorial(n - 1);
        }
    %>
  </body>
</html>
```

JSP scriptlet

JSP expression

JSP declaration



JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the client's request, which is an instance of HttpServletRequest. You can use it to access request parameters, HTTP headers such as cookies, hostname, etc.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the servlet's response, which is an instance of HttpServletResponse. You can use it to set response type and send output to the client.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the character output stream, which is an instance of PrintWriter obtained from response.getWriter(). You can use it to send character content to the client.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the HttpSession object associated with the request, obtained from request.getSession().

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the ServletContext object for storing persistent data for all clients. The difference between session and application is that session is tied to one client, but application is for all clients to share persistent data.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the ServletConfig object for the page.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Represents the PageContext object. PageContext is a new class introduced in JSP to give a central point of access to many page attributes.

JSP Predefined Variables

You can use variables in JSP. For convenience, JSP provides eight predefined variables from the servlet environment that can be used with JSP expressions and scriptlets. These variables are also known as *JSP implicit objects*.

request
response
out
session
application
config
pagecontext
page

Page is an alternative to this.

Example of JSP Predefined Variable

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

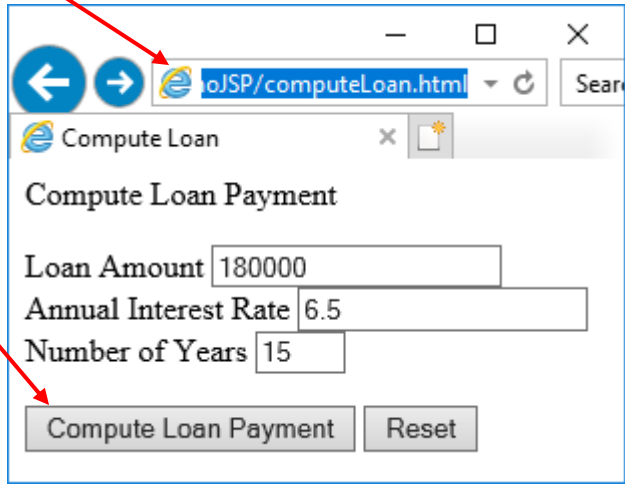
```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

```
<!-- computeLoan.html -->
<html>
<head>
<title>Compute Loan</title>
</head>
<body>
Compute Loan Payment

<form method="get" action="computeLoan.jsp">
<p>Loan Amount
    <input type="text" name="loanAmount"> <br>
Annual Interest Rate
    <input type="text" name="annualInterestRate"> <br>
Number of Years <input type="text" name="numberOfYears" size="3">
</p>
<p><input type="submit" name="Submit" value="Compute Loan Payment">
   <input type="reset" value="Reset"></p>
</form>
</body>
</html>
```

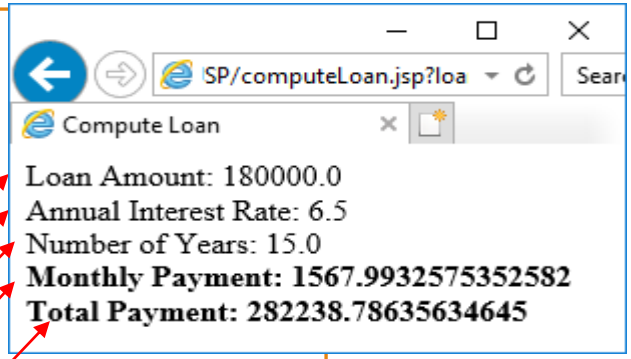
Example 4.2 Computing Loan

Write an HTML page that prompts the user to enter loan amount, annual interest rate, and number of years. Clicking the Compute Loan Payment button invokes a JSP to compute and display the monthly and total loan payment.



```
<!-- computeLoan.jsp -->
<html>
<head>
<title>ComputeLoan</title>
</head>
<body>
<% double loanAmount = Double.parseDouble(
    request.getParameter("loanAmount"));
   double annualInterestRate = Double.parseDouble(
    request.getParameter("annualInterestRate"));
   double numberOfYears = Integer.parseInt(
    request.getParameter("numberOfYears"));
   double monthlyInterestRate = annualInterestRate / 1200;
   double monthlyPayment = loanAmount * monthlyInterestRate /
     (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
   double totalPayment = monthlyPayment * numberOfYears * 12; %>
Loan Amount: <%= loanAmount %><br>
Annual Interest Rate: <%= annualInterestRate %><br>
Number of Years: <%= numberOfYears %><br>
<b>Monthly Payment: <%= monthlyPayment %><br>
Total Payment: <%= totalPayment %><br></b>
</body>
</html>
```

Predefined variable



JSP Directives

A JSP directive is a statement that gives the JSP engine information about the JSP page. For example, if your JSP page uses a Java class from a package other than the java.lang package, you have to use a directive to import this package. The general syntax for a JSP directive is as follows:

```
<%@ directive attribute="value" %>, or

<%@ directive attribute1="value1"
                attribute2="value2"
                ...
                attribute="valuen" %>
```

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page
include
tablib

page lets you provide information for the page, such as importing classes and setting up content type. The page directive can appear anywhere in the JSP file.

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page
include
tablib

include lets you insert a file to the servlet when the page is translated to a servlet. The include directive must be placed where you want the file to be inserted.

Three JSP Directives

Three possible directives are the following: page, include, and tablib.

page
include
tablib

tablib lets you define custom tags.

```
<!-- computeLoanCls.jsp -->
<html>
<head>
<title>Compute Loan Using the Loan Class</title>
</head>
<body>
<%@ page import = "chap4.Loan" %>
<% double loanAmount = Double.parseDouble(
     request.getParameter("loanAmount"));
   double annualInterestRate = Double.parseDouble(
     request.getParameter("annualInterestRate"));
   int numberOfYears = Integer.parseInt(
     request.getParameter("numberOfYears"));
   Loan loan = new Loan(annualInterestRate, numberOfYears,
loanAmount);
%>
Loan Amount: <%= loanAmount %><br>
Annual Interest Rate: <%= annualInterestRate %><br>
Number of Years: <%= numberOfYears %><br>
<b>Monthly Payment: <%= loan.monthlyPayment() %><br>
Total Payment: <%= loan.totalPayment() %><br></b>
</body>
</html>
```

Example: Computing Loan Using the Loan Class

Use the Loan class to simplify Example 4.2. You can create an object of Loan class and use its monthlyPayment() and totalPayment() methods to compute the monthly payment and total payment.

Import a class. The class must be placed in a package (e.g. package chap4)

```
package chap4;

public class Loan {
    private double loanAmount, annualInterestRate;
    private int numberOfYears;

    public Loan(double loanAmount, int numberOfYears, double
annualInterestRate) {
        this.loanAmount = loanAmount;
        this.annualInterestRate = annualInterestRate;
        this.numberOfYears = numberOfYears;
    }

    public double monthlyPayment() {
        double monthlyInterestRate = annualInterestRate / 1200;
        return (loanAmount * monthlyInterestRate) /
             (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
    }

    public double totalPayment(){
        return monthlyPayment() * numberOfYears * 12;
    }
}
```

Example: Computing Loan Using the Loan Class

Use the Loan class to simplify Example 4.2. You can create an object of Loan class and use its monthlyPayment() and totalPayment() methods to compute the monthly payment and total payment.

What is JSTL?

- **JSTL (JSP Standard Tag Libraries)**is a collection of JSP custom tags developed by Java Community Process, www.jcp.org. The reference implementation is developed by the Jakarta project, jakarta.apache.org.

- JSTL allows you to program your JSP pages using tags, rather than the scriptlet code that most JSP programmers are already accustomed to. JSTL can do nearly everything that regular JSP scriptlet code can do.

JSTL Tags (Custom Actions)

- Can Define Your own!
- Description
  - Define
  - Install
  - Declare
  - Use

JSTL Tags

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
    ...
    <c:out value="${1 + 2 + 3}" />
```

Note: the pattern ${ } is declared that identifies EL expressions. In short EL expressions are enclosed by the ${ } characters.
The dollar symbol is just for printing the value of the variable. Its just a short cut for a scriplet that prints . It relies on Java Bean specs and it is same as ".get"

JSP Standard Tag Library

- Built on custom tag infrastructure

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html>
  <head>
    <title>JSP is Easy</title>
  </head>
  <body bgcolor="white">
    <h1>JSP is as easy as ...</h1>
     1 + 2 + 3 = <c:out value="${1 + 2 + 3}" />
  </body>
</html>
```

JSTL Control Tags

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c "uri="http://java.sun.com/jstl/core" %>

  <c:if test="${2>0}">
       It's true that (2>0)!
  </c:if>

  <c:forEach items="${paramValues.food}" var="current">
       <c:out value="${current}" /> 
  </c:forEach>
```

COUNT TO TEN EXAMPLE USING SCRIPTLET:

JSTL was introduced to allow JSP programmers to program using tags rather than Java code.
To show why this is preferable, a quick example is in order. We will examine a very simple JSP page that counts to ten.
We will examine this page both as regular scriptlet-based JSP, and then as JSTL. When the count to ten example is programmed using scriptlet based JSP, the JSP page appears as follows:

```
<html>
<head>
  <title>Count to 10 in JSP scriptlet</title>
</head>
<body>
    <%for(int i=1;i<=10;i++) {%>
        <%=i%> <br/>
    <%}%>
</body>
</html>
```

COUNT TO TEN EXAMPLE USING JSTL:

The following code shows how the count to ten example would be written using JSTL. As you can see, this code listing is much more constant, as only tags are used. HTML and JSTL tags are mixed to produced the example.

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<html>
   <head>
    <title>Count to 10 Example (using JSTL)</title>
  </head>
  <body><c:forEach var="i" begin="1" end="10" step="1">
      <c:out value="${i}" />

     <br/>
      </c:forEach>
   </body>
</html>
```

The JSTL Tag Libraries

- The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:
  - **Core Tags**
  - **Formatting tags**
  - **SQL tags**
  - **XML tags**
  - **JSTL Functions**
- https://www.javatpoint.com/jstl

CoreTags

- The core group of tags are the most frequently used JSTL tags. Following is the syntax to include JSTL Core library in your JSP:
  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
- There are following Core JSTL Tags:

Tag	Description
<c:out >	Like <%= ... >, but for expressions.
<c:set >	Sets the result of an expression evaluation in a 'scope'
<c:remove >	Removes a scoped variable (from a particular scope, if specified).
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evalutes its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<c:when>	Subtag of <choose> that includes its body if its condition evalutes to 'true'.
<c:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.
<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
<c:forEach >	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .
<c:forTokens>	Iterates over tokens, separated by the supplied delimeters.
<c:param>	Adds a parameter to a containing 'import' tag's URL.
<c:redirect >	Redirects to a new URL.
<c:url>	Creates a URL with optional query parameters

Formatting tags

- The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites. Following is the syntax to include Formatting library in your JSP:
  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
- There are following Formatting JSTL Tags:

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern
<fmt:parseDate>	Parses the string representation of a date and/or time
<fmt:bundle>	Loads a resource bundle to be used by its tag body.
<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding

SQL tags

- The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server. Following is the syntax to include JSTL SQL library in your JSP:
- **<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>**
- There are following SQL JSTL Tags:

Tag	Description
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
<sql:param>	Sets a parameter in an SQL statement to the specified value.
<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<sql:transaction >	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

XML tags

- The JSTL XML tags provide a JSP-centric way of creating and manipulating XML documents. Following is the syntax to include JSTL XML library in your JSP:

  <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
- The JSTL XML tag library has custom tags for interacting with XML data. This includes parsing XML, transforming XML data, and flow control based on XPath expressions.

- Before you proceed with the examples, you would need to copy following two XML and XPath related libraries into your <Tomcat Installation Directory>\lib:

  - **XercesImpl.jar:** Download it from http://www.apache.org/dist/xerces/j/
  - **xalan.jar:** Download it from http://xml.apache.org/xalan-j/index.html

XML tags

- There are following XML JSTL Tags:

Tag	Description
<x:out>	Like <%= ... >, but for XPath expressions.
<x:parse>	Use to parse XML data specified either via an attribute or in the tag body.
<x:set >	Sets a variable to the value of an XPath expression.
<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
<x:forEach>	To loop over nodes in an XML document.
<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<x:when >	Subtag of <choose> that includes its body if its expression evalutes to 'true'
<x:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<x:transform >	Applies an XSL transformation on a XML document
<x:param >	Use along with the transform tag to set a parameter in

Conclusion

- Java Server Pages (JSP) lets you separate the dynamic part of your pages from the static HTML.
- One can simply write the regular HTML in the normal manner, using whatever Web-page-building tools you normally use.
- One can enclose then the code for the dynamic parts in special tags, most of which

  start with "<%"
  and end with "%>"

References

- https://www.javatpoint.com/jsp-tutorial