CSC584 Enterprise Programming

Chapter 4 – JSP

MARSHIMA MOHD ROSLI

COMPUTER SCIENCE

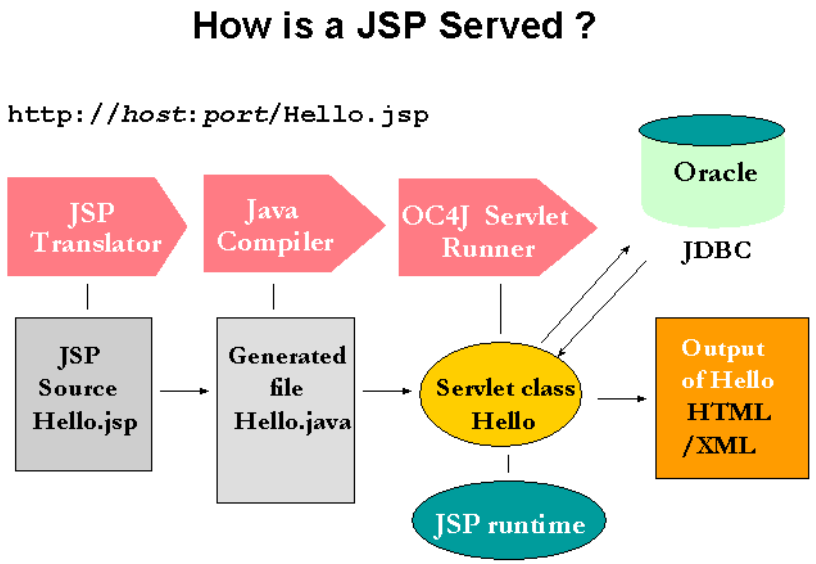
Chapter 4 Outline

JSP

- 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
- Compiled at Request time
  - (first request, a little slow)
- Execution: Request → JSP Servlet's service method



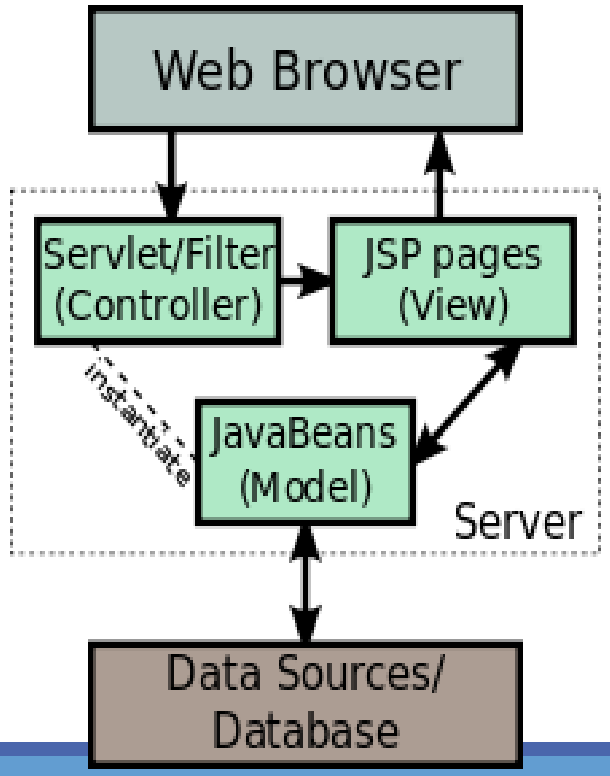
Example:JSP

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

JavaServer Pages Technology

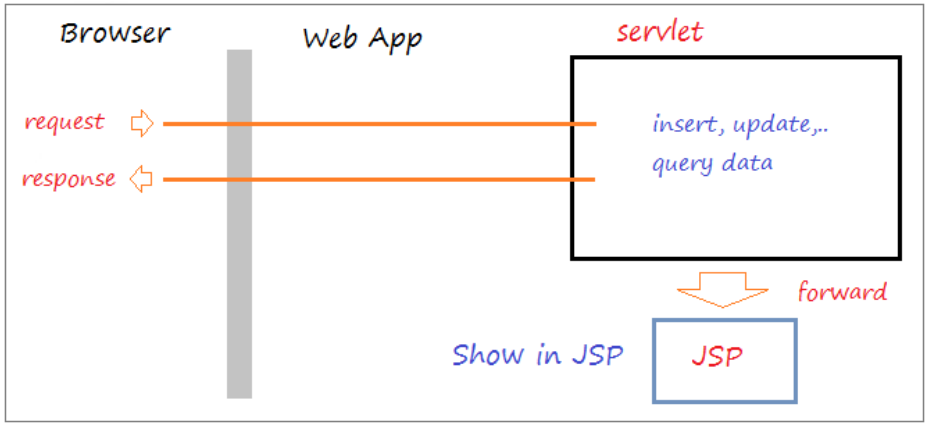
- JavaServer Pages (JSP) technology provides a simplified, fast way to create web pages that display dynamically-generated content.

- **JSP is a standard extension of Java and is defined on top of Servlet extensions.**
- **Its goal is to simplify management and creation of dynamic web pages.**
- **It is platform-independent, secure, and it makes use of Java as a server side scripting language.**

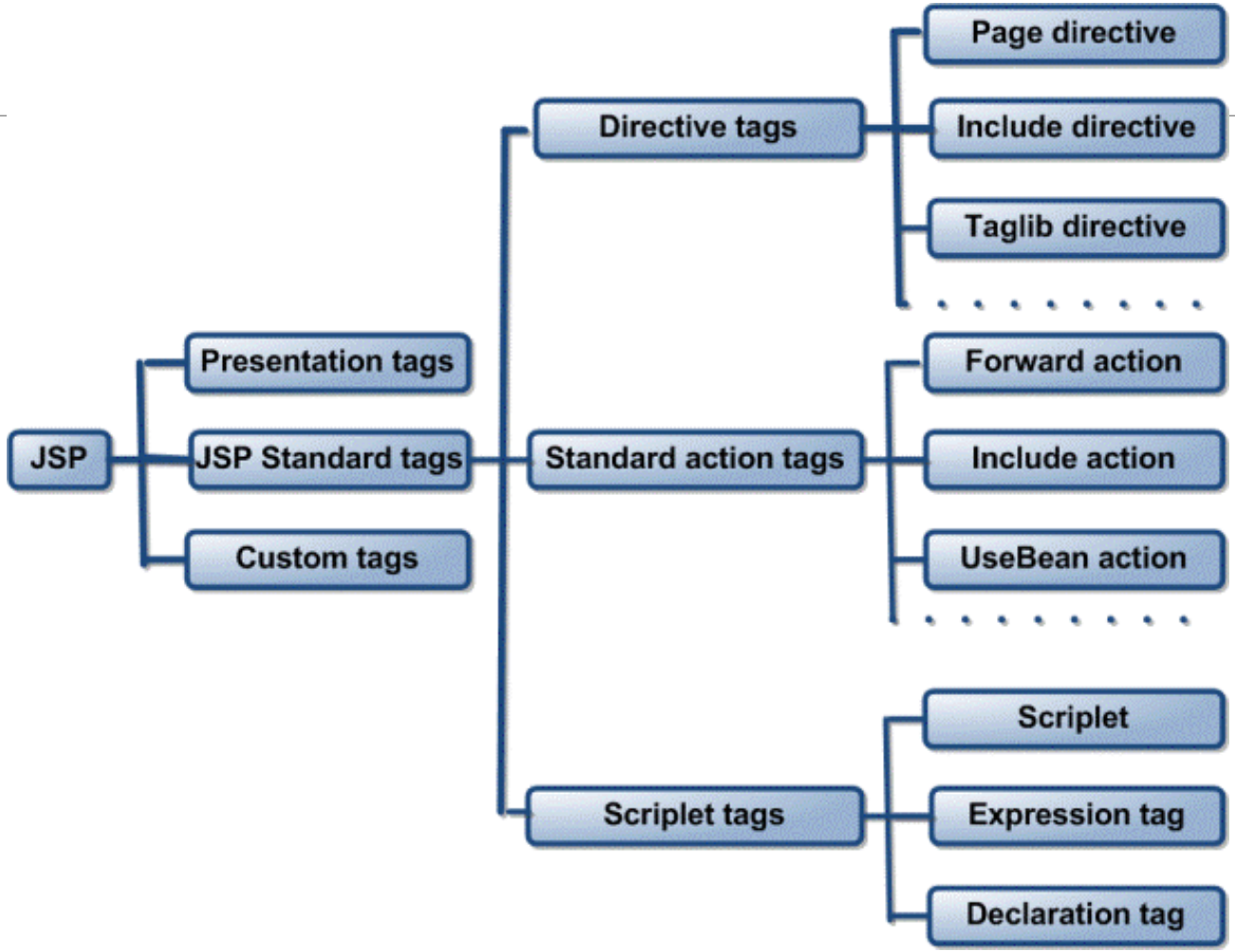


JSP Page

- A JSP page is a page created by the web developer that includes JSP technology-specific tags, *declarations, and possibly scriptlets*, in combination with other *static HTML or XML tags*.
- A JSP page has the extension *.jsp*; this signals to the web server that the *JSP engine* will process elements on this page.



What are the different types of JSP tags?



Advantages

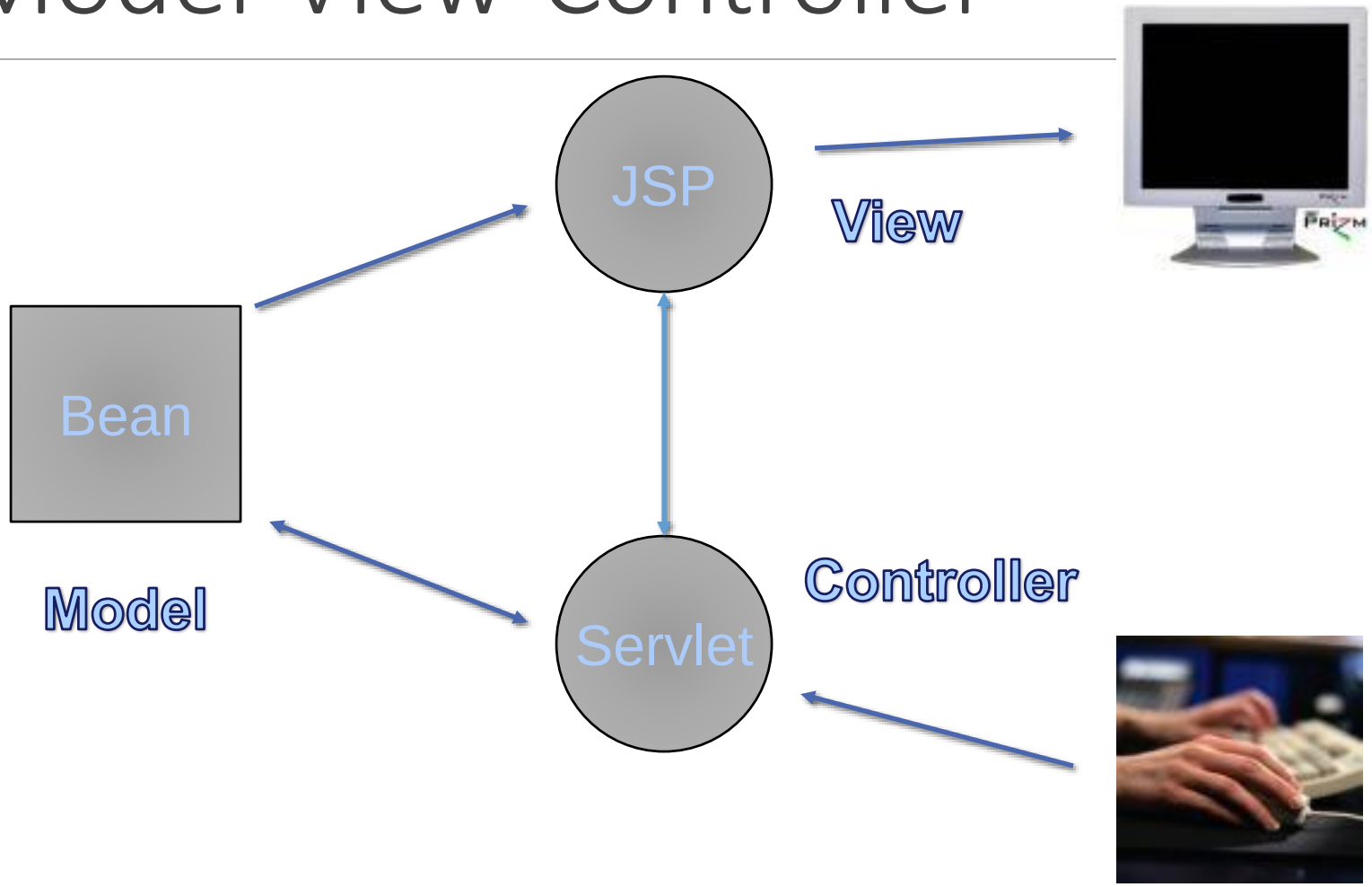
Code -- Computation

HTML -- Presentation

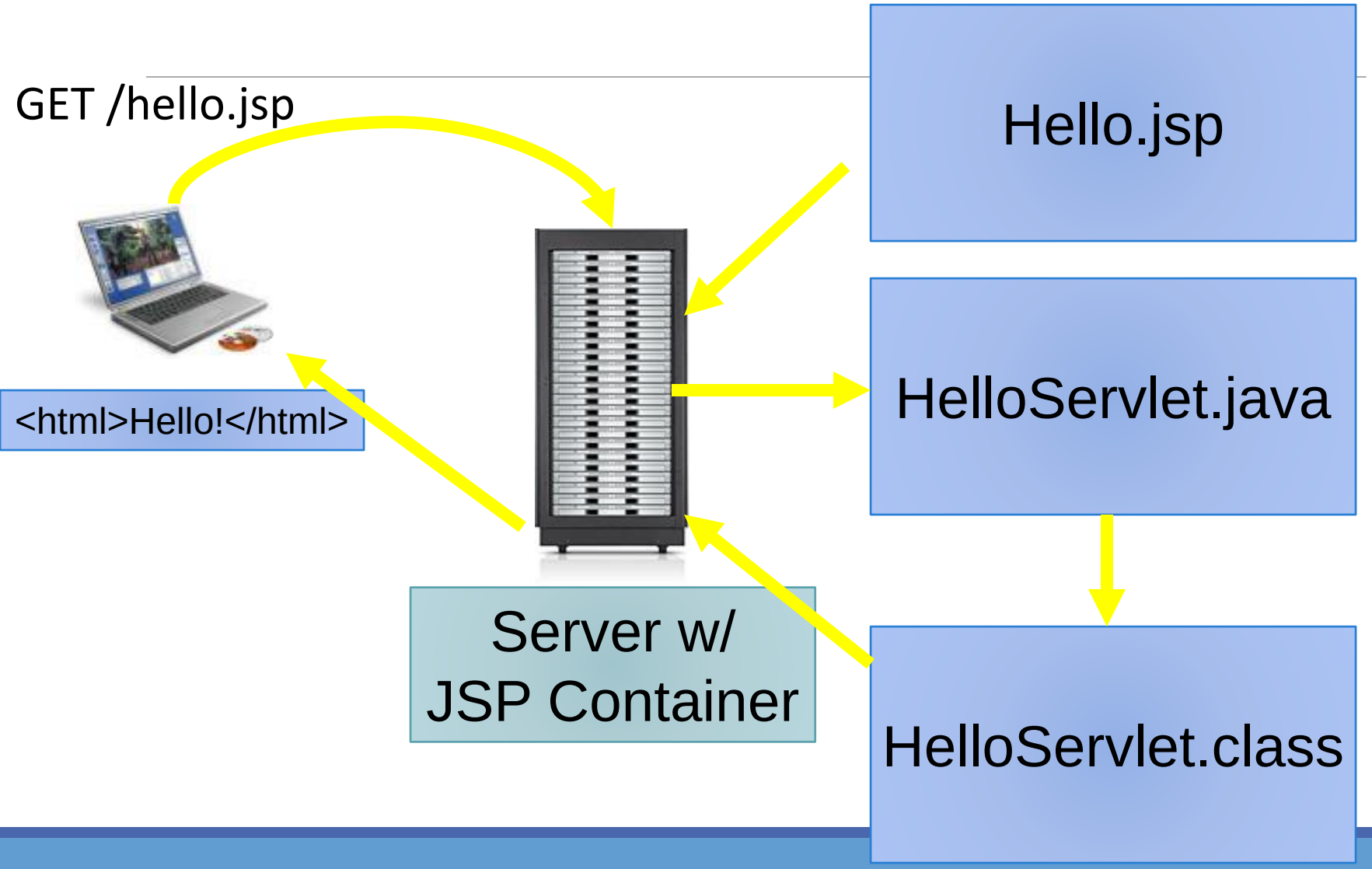
Separation of Roles

- Developers
- Content Authors/Graphic Designers/Web Masters

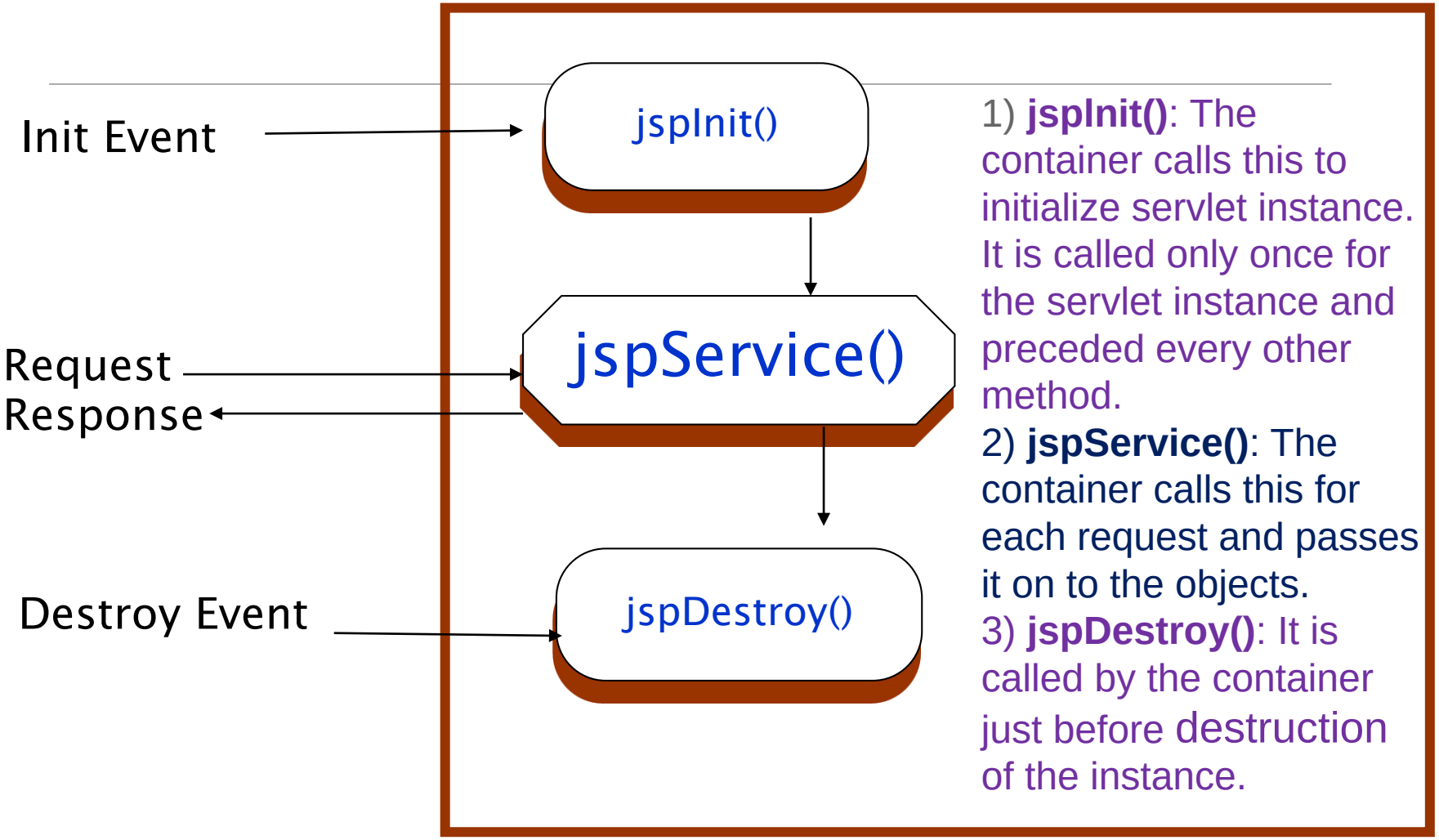
Model-View-Controller



JSP Big Picture



JSP Lifecycle



1) **jspInit()**: The container calls this to initialize servlet instance. It is called only once for the servlet instance and preceded every other method.
2) **jspService()**: The container calls this for each request and passes it on to the objects.
3) **jspDestroy()**: It is called by the container just before destruction of the instance.

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:
  <ul>
   <li>User Name:
```
template text

```
    <jsp:getProperty name="userInfo"
     property="userName"/>
```
JSP element

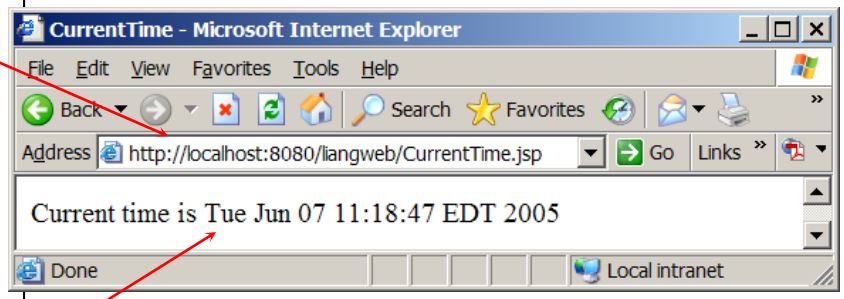
```
   <li>Email Address:
```
template text

```
    <jsp:getProperty name="userInfo"
    property="emailAddr"/>
```
JSP element

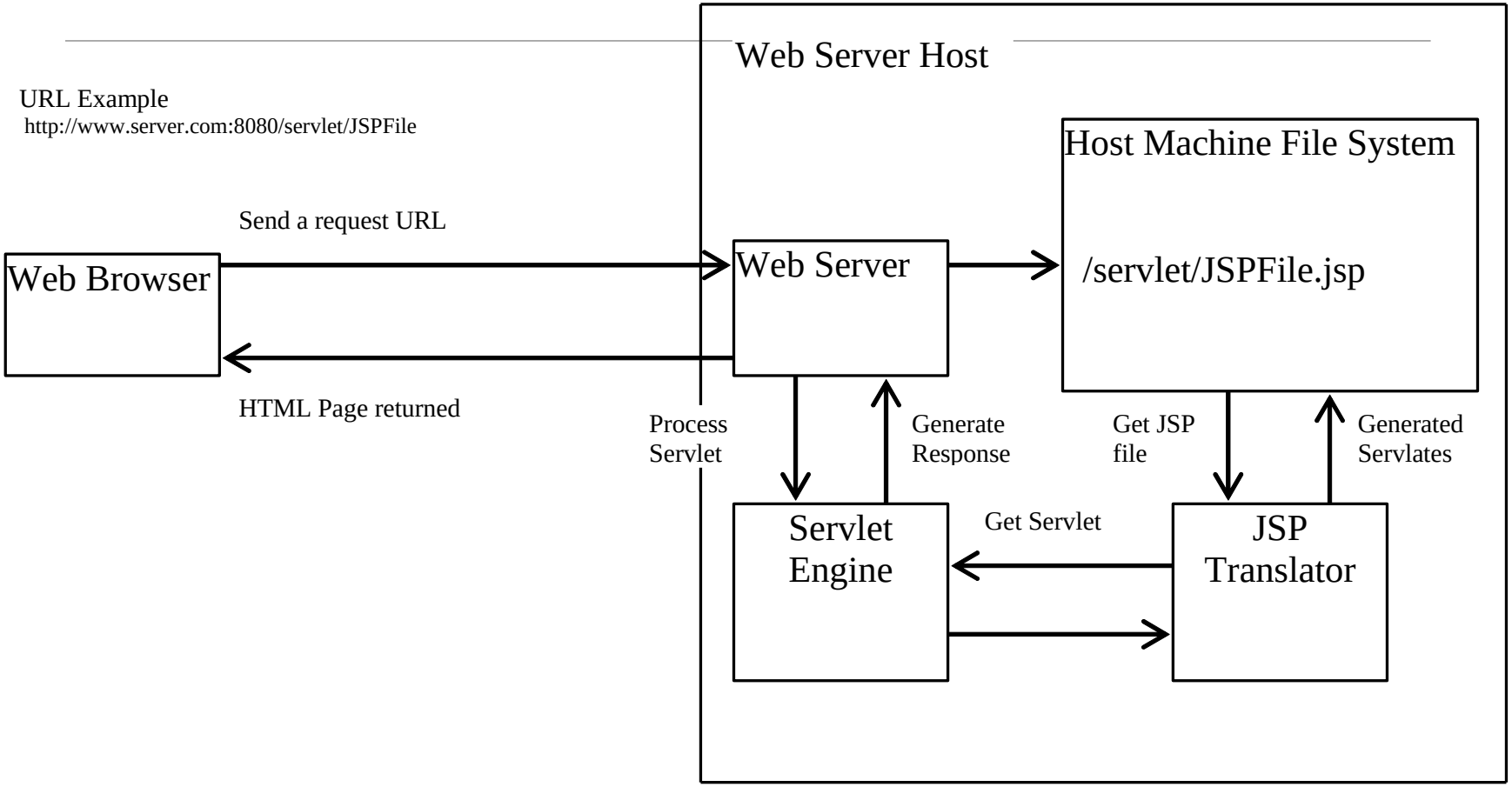
```
  </ul>
 </body>
</html>
```
template text

A Simple JSP

```
<!-- CurrentTime.jsp -->

<HTML>

<HEAD>

<TITLE>

CurrentTime

</TITLE>

</HEAD>

<BODY>

Current time is <%= new java.util.Date() %>

</BODY>

</HTML>
```



How Is a JSP Processed?



Activity

Explain the difference between Servlet and JSP.

JSP Constructs

There are three types of scripting constructs you can use to insert Java code into the result 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 %>

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 --%>

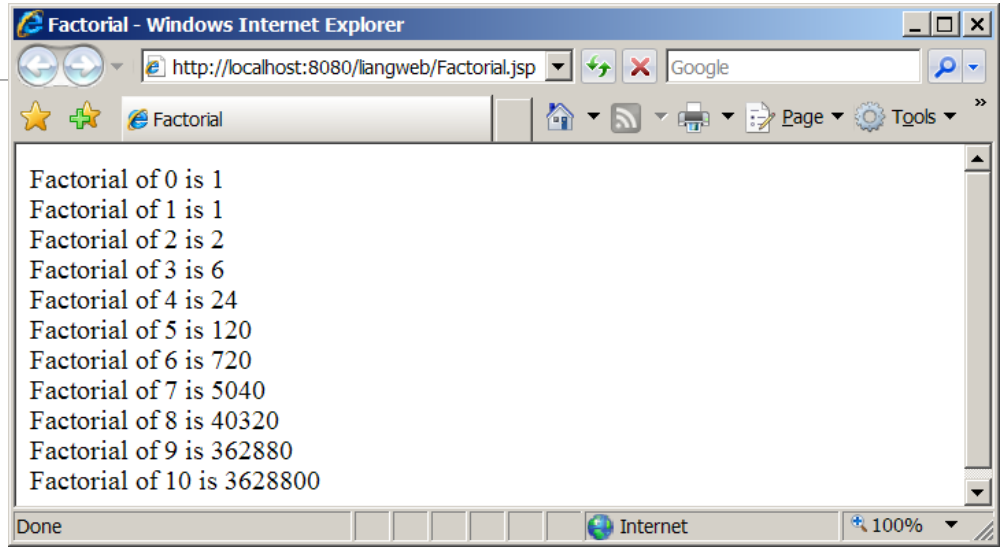
Example 40.1 Computing Factorials

```
<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



Activity

Give examples

- **Expressions**
  - Format: <%= expression %>
  - Evaluated and inserted into the servlet's output. I.e., results in something like out.print(expression)
- **Scriptlets**
  - Format: <% code %>
  - Inserted verbatim into the servlet's _jspService method (called by service)
- **Declarations**
  - Format: <%! code %>
  - Inserted verbatim into the body of the servlet class, outside of any existing methods

JSP Predefined Variables

UNIVERSITI TEKNOLOGI MARA

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

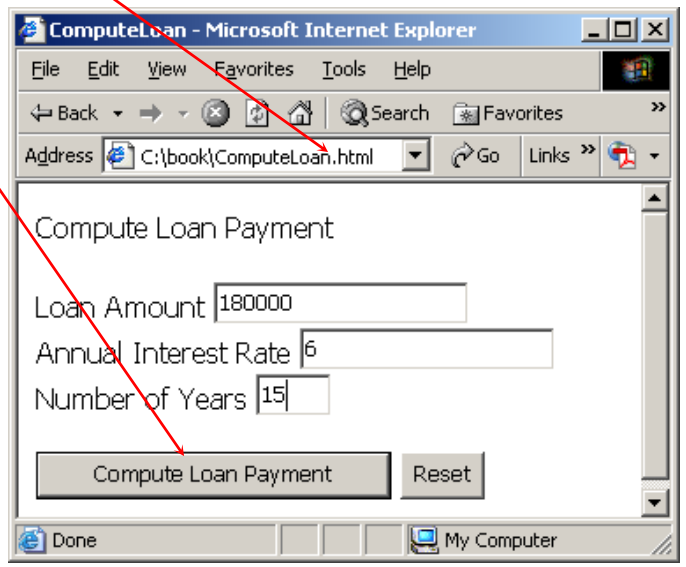
page

Page is an alternative to this.

Example 40.2
Computing Loan

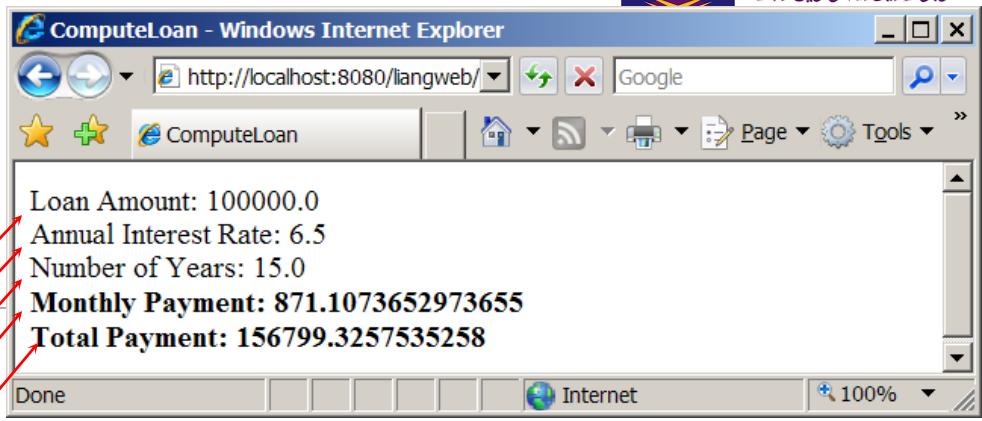
Write an HTML page that prompts the user to enter loan amount, annual interest rate, and number of years. Clicking the Compute Loan Payment button invokes a JSP to compute and display the monthly and total loan payment.

```
<!-- ComputeLoan.html -->
<html>
<head>
<title>ComputeLoan</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>
```



```
<!-- 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="vlauen" %>
```

Three JSP Directives

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

page

include

taglib

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 taglib

page

include

taglib

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 taglib

page

include

taglib

tablib lets you define custom tags.

Example: Computing Loan Using the Loan Class

```
<!-- ComputeLoan.jsp -->
<html>
<head>
<title>ComputeLoan Using the Loan Class</title>
</head>
<body>
<%@ page import = "chapter40.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>
```

Use the Loan class to simplify Example 40.2. 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 chapter40.

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

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

JSP Standard Tag Library

- Built on custom tag infrastructure

```
<%@ page contentType="text/html" %>


<html>
  <head>
    <title>JSP is Easy</title>
  </head>
  <body bgcolor="white">
    <h1>JSP is as easy as ...</h1>
  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 produce the example.

```
<%@ taglib uri="http://java.sun.com/jstl/core" 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>
```

COUNT TO TEN EXAMPLE USING JSTL:

- When you examine the preceding source code, you can see that the JSP page consists entirely of tags.

- The code makes use of HTML tags such as <head> and
. The use of tags is not confined just to HTML tags. We can modify that code to print numbers from 1 to n. The code is given in the next slide.

- This code also makes use of JSTL tags such as <c:forEach> and <c:out>. In this article you will be introduced to some of the basics of JSTL.

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**

Core Tags:

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" %>
```

Following is the list of 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" %>

Following is the list of 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

Following is the list of 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 the XSLT stylesheet

JSTL Functions:

UNIVERSITI TEKNOLOGI MARA

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP:

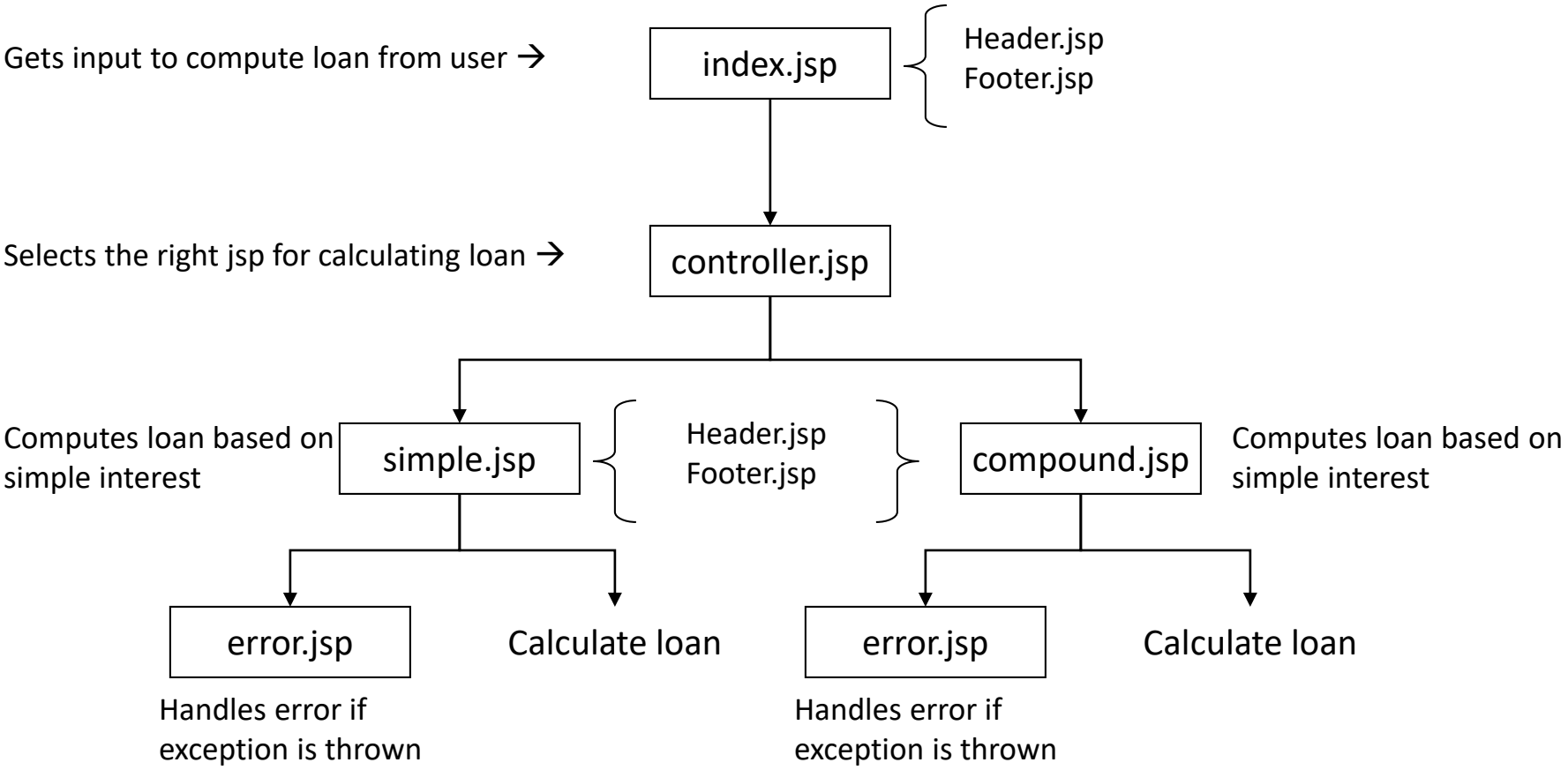
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

Following is the list of JSTL Functions:

Function	Description
fn:contains()	Tests if an input string contains the specified substring.
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
fn:endsWith()	Tests if an input string ends with the specified suffix.
fn:escapeXml()	Escapes characters that could be interpreted as XML markup.
fn:indexOf()	Returns the index within a string of the first occurrence of a specified substring.
fn:join()	Joins all elements of an array into a string.
fn:length()	Returns the number of items in a collection, or the number of characters in a string.
fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
fn:split()	Splits a string into an array of substrings.

Example

Loan Calculator



Loan Calculator

index.jsp

```
<html>
  <head>
    <title>Include</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <form action="controller.jsp">
      <table border="0" style="font-family:verdana;font-size:10pt;">
        <tr>
          <td>Amount:</td>
          <td><input type="text" name="amount" />
        </tr>
        <tr>
          <td>Interest in %:</td>
          <td><input type="text" name="interest"/></td>
        </tr>
        <tr>
          <td>Compound:</td>
          <td><input type="radio" name="type" value="C" checked/></td>
        </tr>
        <tr>
          <td>Simple:</td>
          <td><input type="radio" name="type" value="S" /></td>
        </tr>
        <tr>
          <td>Period:</td>
          <td><input type="text" name="period"/></td>
        </tr>
      </table>
      <input type="submit" value="Calculate"/>
    </form>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Loan Calculator

Miscelaneous

controller.jsp

```
<%
  String type = request.getParameter("type");
  if(type.equals("S")) {
%>
<jsp:forward page="/simple.jsp"/>
<%
  } else {
%>
 <jsp:forward page="/compound.jsp"/>
<%
  }
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
  <head>
    <title>Simple</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <p style="color=#FF0000"><b><%=
          exception.getMessage() %></b></p>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

header.jsp

```
<h3>Loan Calculator</h3>
```

footer.jsp

```
<%= new java.util.Date() %>
```

Loan Calculator

simple.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
  public double calculate(double amount, double interest,
          int period) {
    if(amount <= 0) {
      throw new IllegalArgumentException("Amount should
          be greater than 0: " + amount);
    }
    if(interest <= 0) {
      throw new IllegalArgumentException("Interest should
          be greater than 0: " + interest);
    }
    if(period <= 0) {
      throw new IllegalArgumentException("Period should
          be greater than 0: " + period);
    }
    return amount*(1 + period*interest/100);
  }
%>
```

```
<html>
  <head>
    <title>Simple</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <%
      double amount =
          Double.parseDouble(request.getParameter("amount")
          );
      double interest =
          Double.parseDouble(request.getParameter("interest")
          );
      int period =
          Integer.parseInt(request.getParameter("period"));
    %>
    <b>Pincipal using simple interest:</b>
    <%= calculate(amount, interest, period) %>
    <br/><br/>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Loan Calculator

compound.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
  public double calculate(double amount, double interest,
          int period) {
    if(amount <= 0) {
      throw new IllegalArgumentException("Amount should
          be greater than 0: " + amount);
    }
    if(interest <= 0) {
      throw new IllegalArgumentException("Interest should
          be greater than 0: " + interest);
    }
    if(period <= 0) {
      throw new IllegalArgumentException("Period should
          be greater than 0: " + period);
    }
    return amount*Math.pow(1 + interest/100, period);
  }
%>
```

```
<html>
  <head>
    <title>Compound</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <%
      double amount =
          Double.parseDouble(request.getParameter("amount
          "));
      double interest =
          Double.parseDouble(request.getParameter("interest
          "));
      int period =
          Integer.parseInt(request.getParameter("period"));
    %>
    <b>Pincipal using compound interest:</b>
    <%= calculate(amount, interest, period) %>
    <br/><br/>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Conclusion

JavaServer Pages (JSP) lets you separate the dynamic part of your pages from the static HTML.

1. One can simply write the regular HTML in the normal manner, using whatever Web-page-building tools you normally use.
2. One can enclose then the code for the dynamic parts in special tags, most of which

   start with "<%"

   and end with "%>"

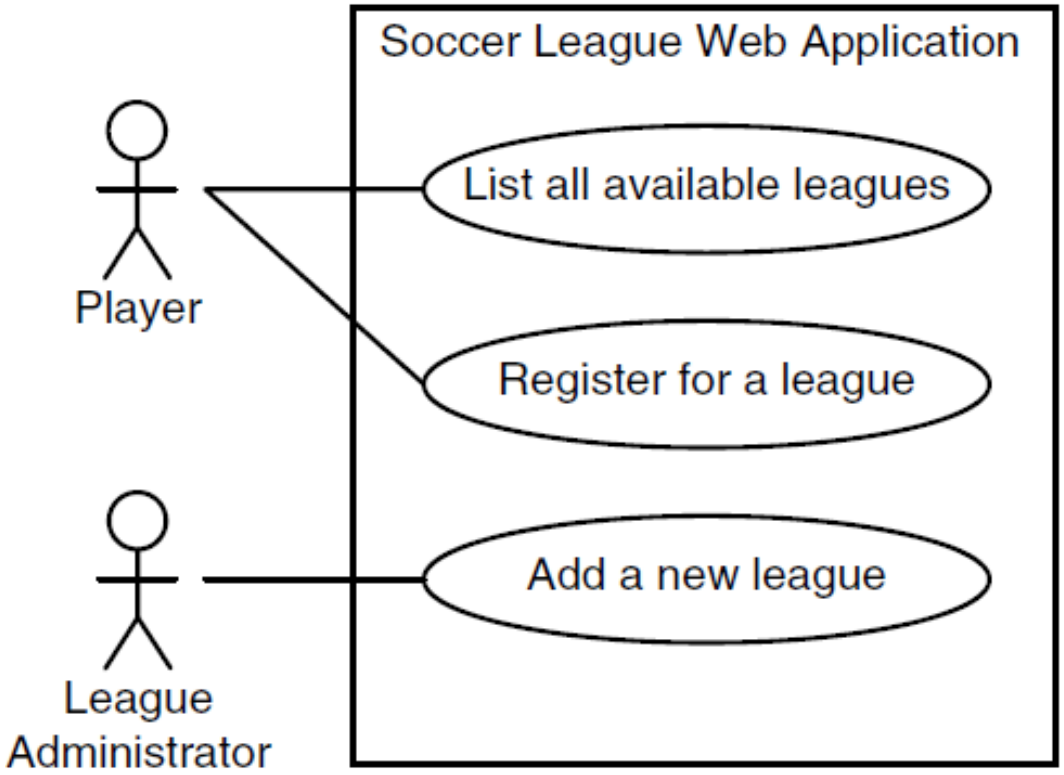
Review JSP

Open kahoot..

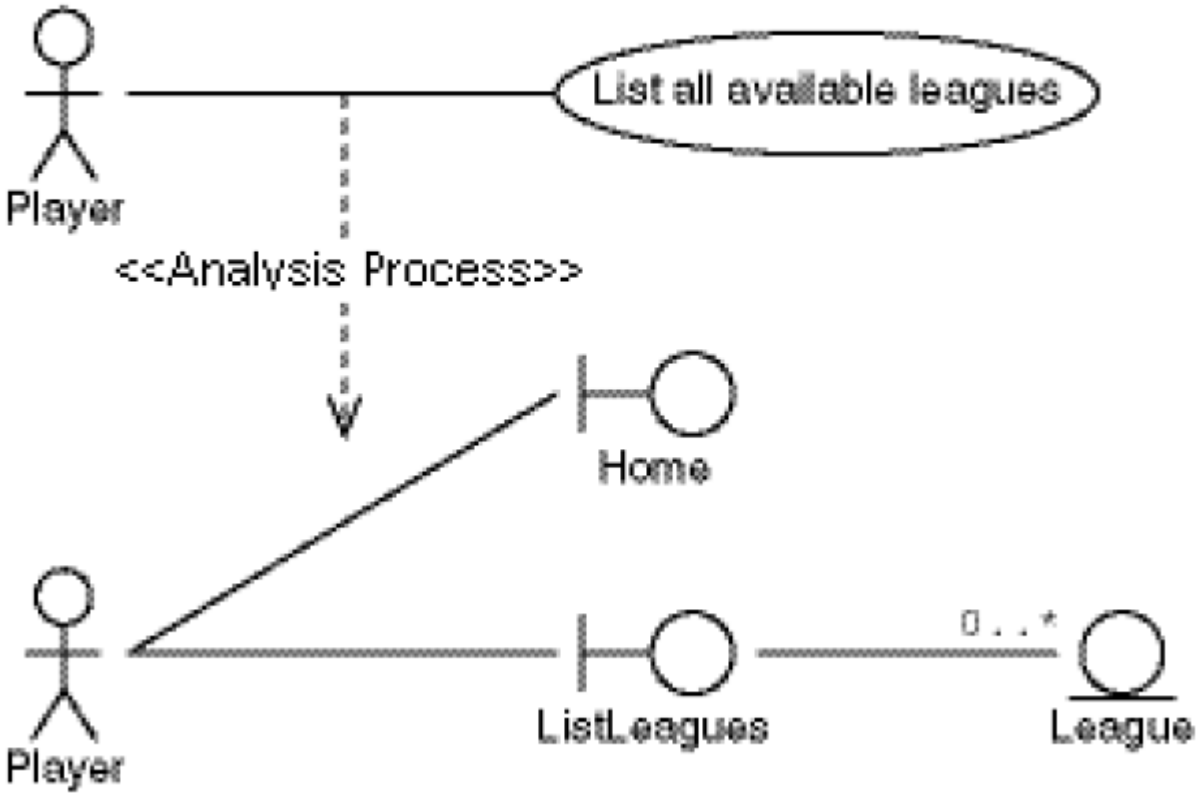
Developing a View Component

- Design a view component
- Develop a simple HTTP servlet
- Configure and deploy a servlet

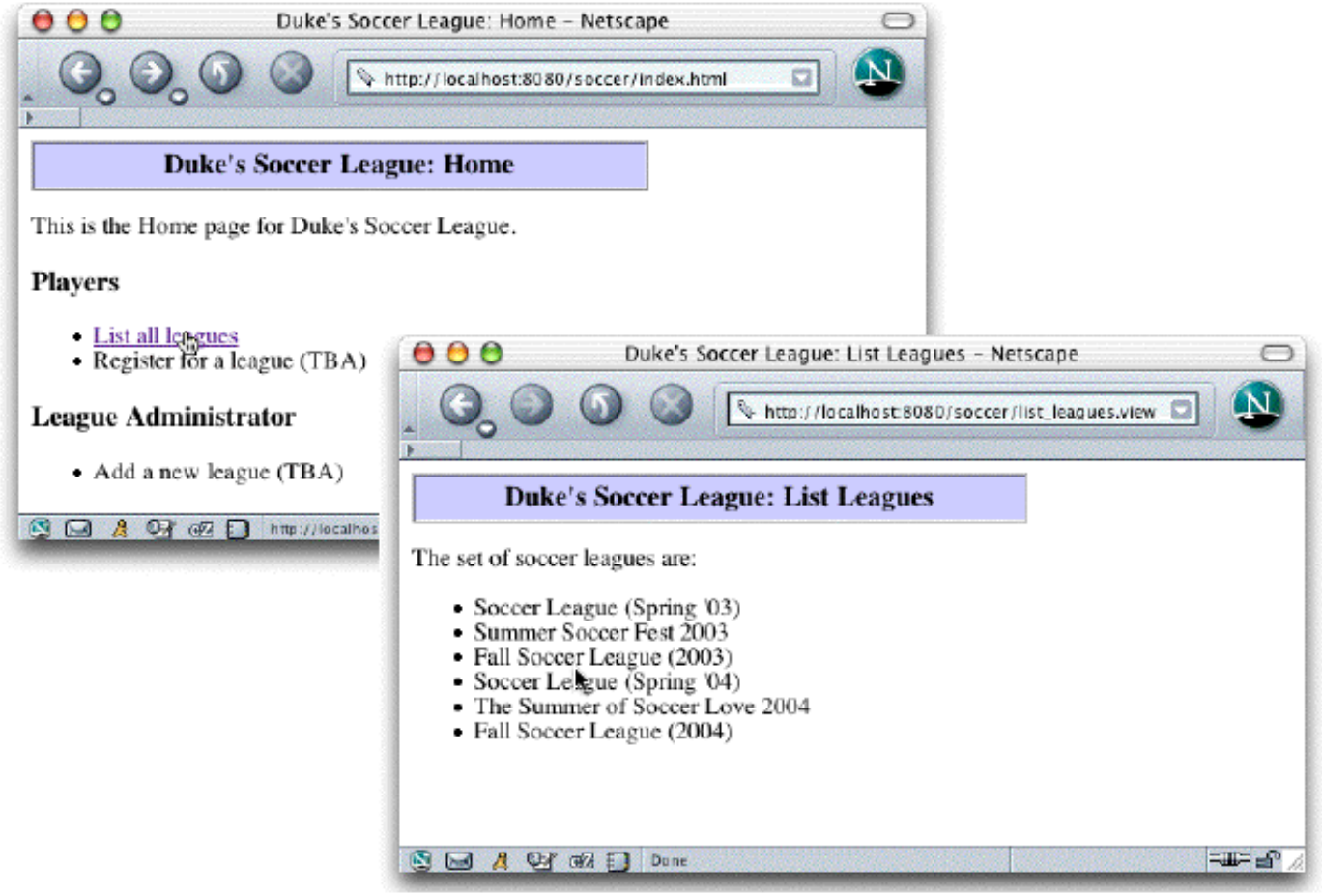
Soccer League Case Study



List Leagues Analysis Model



List Leagues Page Flow



Home Page HTML

```
1   <html>
2
3   <head>
4   <title>Duke's Soccer League: Home</title>
5   </head>
6
7   <body bgcolor='white'>
8
9   <!-- Page Heading -->
10  <table border='1' cellpadding='5' cellspacing='0' width='400'>
11  <tr bgcolor='#CCCCFF' align='center' valign='center' height='20'>
12    <td><h3>Duke's Soccer League: Home</h3></td>
13  </tr>
14  </table>
15
```

Home Page HTML (Part 2)

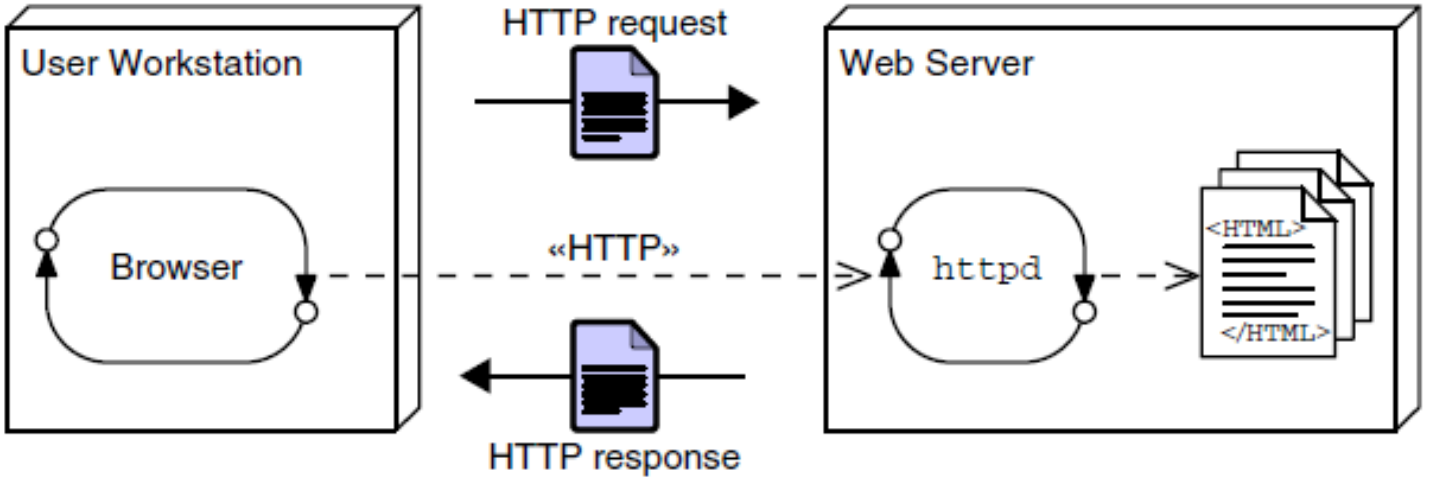
```
16  <p>
17  This is the Home page for Duke's Soccer League.
18  </p>
19
20  <h3>Players</h3>
21
22  <ul>
23    <li><a href='list_leagues.view'>List all leagues</a></li>
24    <li>Register for a league (TBA)</li>
25  </ul>
26
27  <h3>League Administrator</h3>
28
29  <ul>
30    <li>Add a new league (TBA)</li>
31  </ul>
32
33  </body>
34
35  </html>
```

List Leagues Page HTML

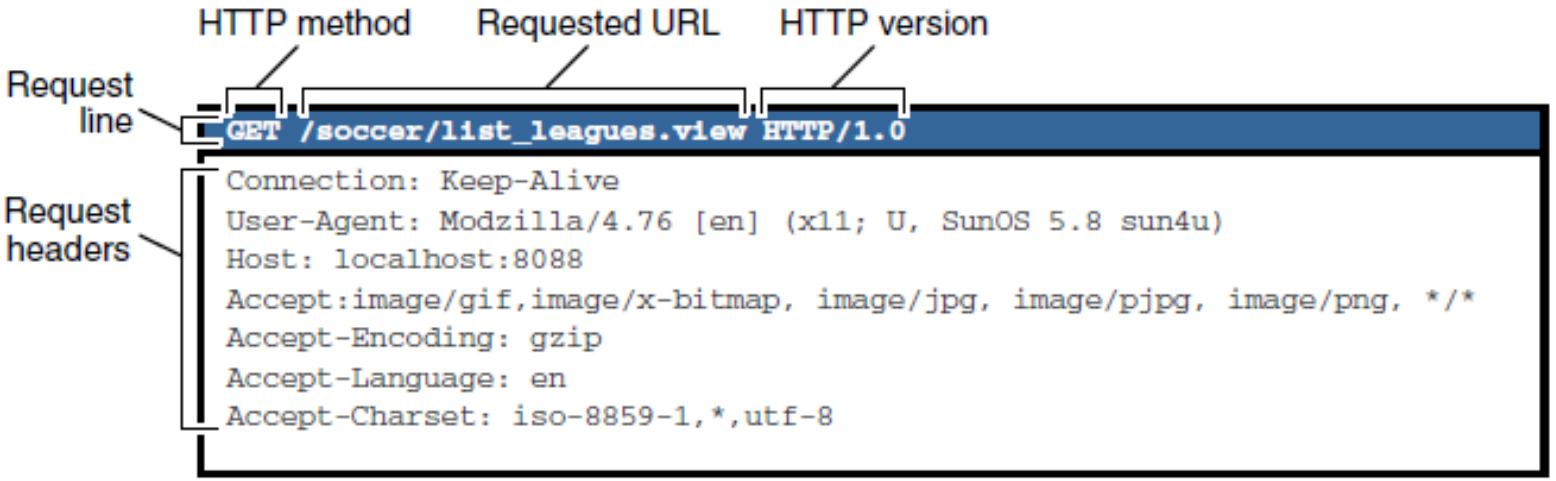
```
<!-- Page Heading -->
<table border='1' cellpadding='5' cellspacing='0' width='400'>
<tr bgcolor='#CCCCFF' align='center' valign='center' height='20'>
  <td><h3>Duke's Soccer League: List Leagues</h3></td>
</tr>
</table>

<p>
The set of soccer leagues are:
</p>

<ul>
  <li>The Summer of Soccer Love 2004</li>
  <li>Fall Soccer League (2003)</li>
  <li>Fall Soccer League (2004)</li>
  <li>Soccer League (Spring '03)</li>
  <li>Summer Soccer Fest 2003</li>
  <li>Soccer League (Spring '04)</li>
</ul>

</body>
</html>
```

Hypertext Transfer Protocol

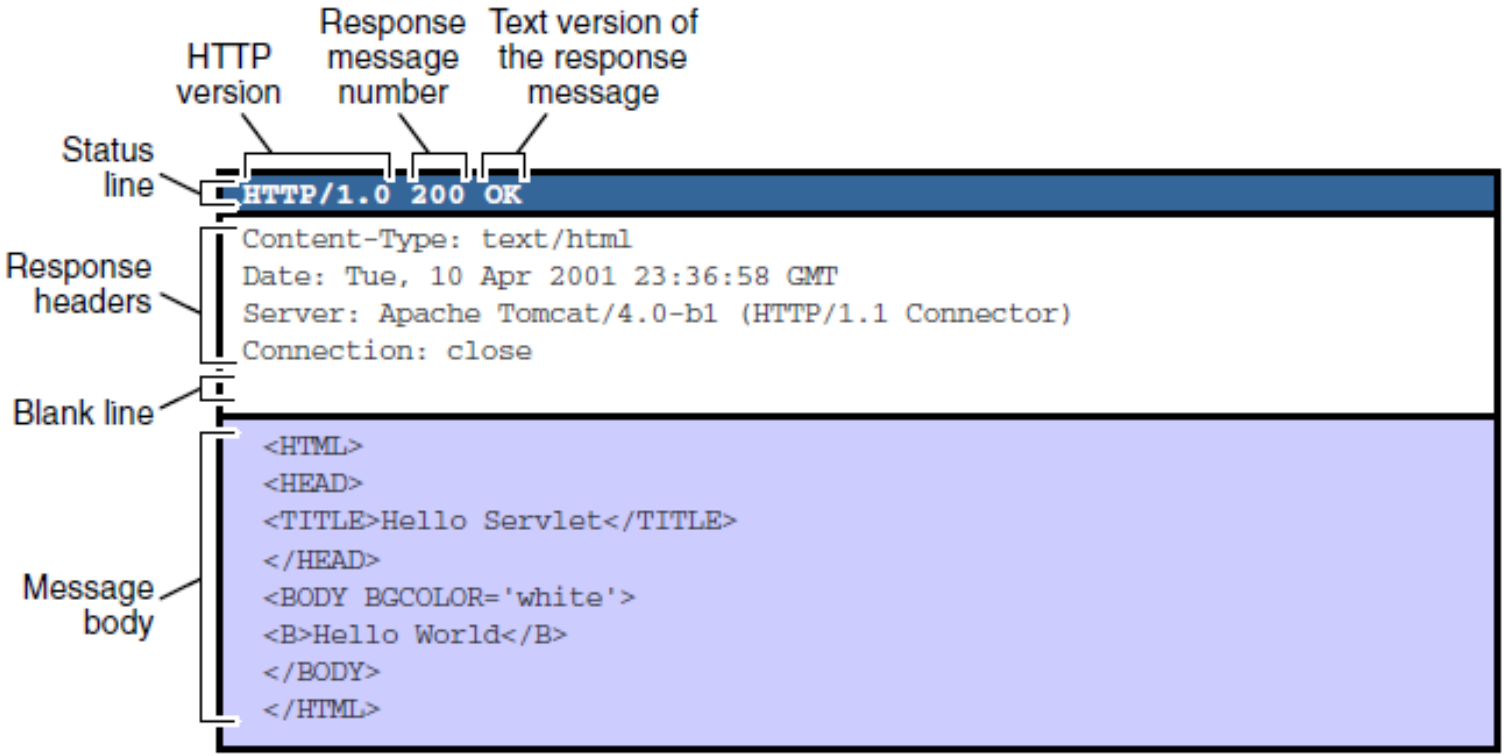
The HTTP client sends a single request to the HTTP daemon (`httpd`) and responds with the requested resource.



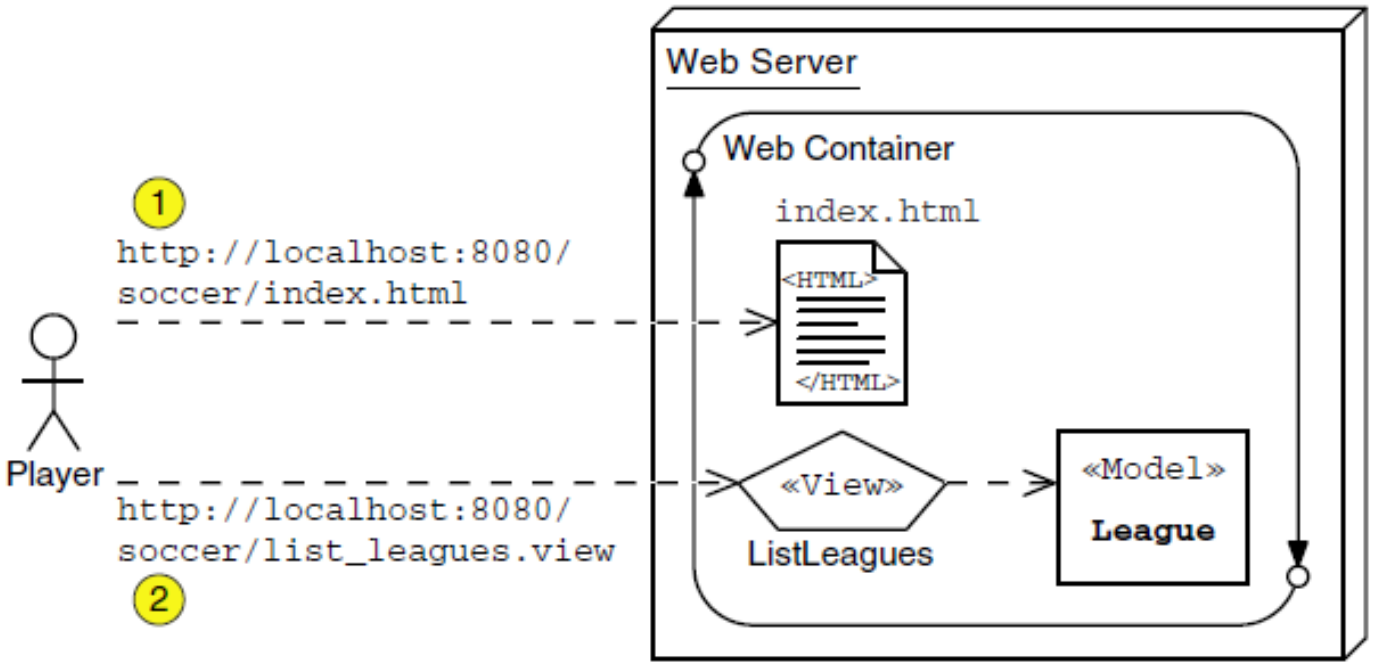
HTTP Request



HTTP Response



List Leagues Architecture Model



The ListLeaguesServlet Code

```
1   package sl314.view;
2
3   import javax.servlet.http.HttpServlet;
4   import javax.servlet.http.HttpServletRequest;
5   import javax.servlet.http.HttpServletResponse;
6   // Support classes
7   import java.io.IOException;
8   import java.io.PrintWriter;
9   // Model classes
10  import sl314.model.League;
11  import java.util.List;
12  import java.util.LinkedList;
13  import java.util.Iterator;
14
15  public class ListLeaguesServlet extends HttpServlet {
16
17    private List leagueList = null;
18
19    public void doGet(HttpServletRequest request,
20                      HttpServletResponse response)
21          throws IOException {
```

The ListLeaguesServlet Code (Part 2)

```
15  public class ListLeaguesServlet extends HttpServlet {
16
17    private List leagueList = null;
18
19    public void doGet(HttpServletRequest request,
20                      HttpServletResponse response)
21          throws IOException {
22
23      // Create the set of leagues
24      leagueList = new LinkedList();
25      leagueList.add( new League(2003, "Spring",
26                                 "Soccer League (Spring '03)") );
27      leagueList.add( new League(2003, "Summer",
28                                 "Summer Soccer Fest 2003") );
29      leagueList.add( new League(2003, "Fall",
30                                 "Fall Soccer League (2003)") );
31      leagueList.add( new League(2004, "Spring",
32                                 "Soccer League (Spring '04)") );
33      leagueList.add( new League(2004, "Summer",
34                                 "The Summer of Soccer Love 2004") );
35      leagueList.add( new League(2004, "Fall",
36                                 "Fall Soccer League (2004)") );
```

The ListLeaguesServlet Code (Part 3)

```
37
38      // Set page title
39      String pageTitle = "Duke's Soccer League: List Leagues";
40
41      // Specify the content type is HTML
42      response.setContentType("text/html");
43      PrintWriter out = response.getWriter();
44
45      // Generate the HTML response
46      out.println("<html>");
47      out.println("<head>");
48      out.println("  <title>" + pageTitle + "</title>");
49      out.println("</head>");
50      out.println("<body bgcolor='white'>");
51
52      // Generate page heading
53      out.println("<!-- Page Heading -->");
54      out.println("<table border='1' cellpadding='5' cellspacing='0'
width='400'>");
55      out.println("<tr bgcolor='#CCCCFF' align='center' valign='center'
height='20'>");
```

The ListLeaguesServlet Code (Part 4)

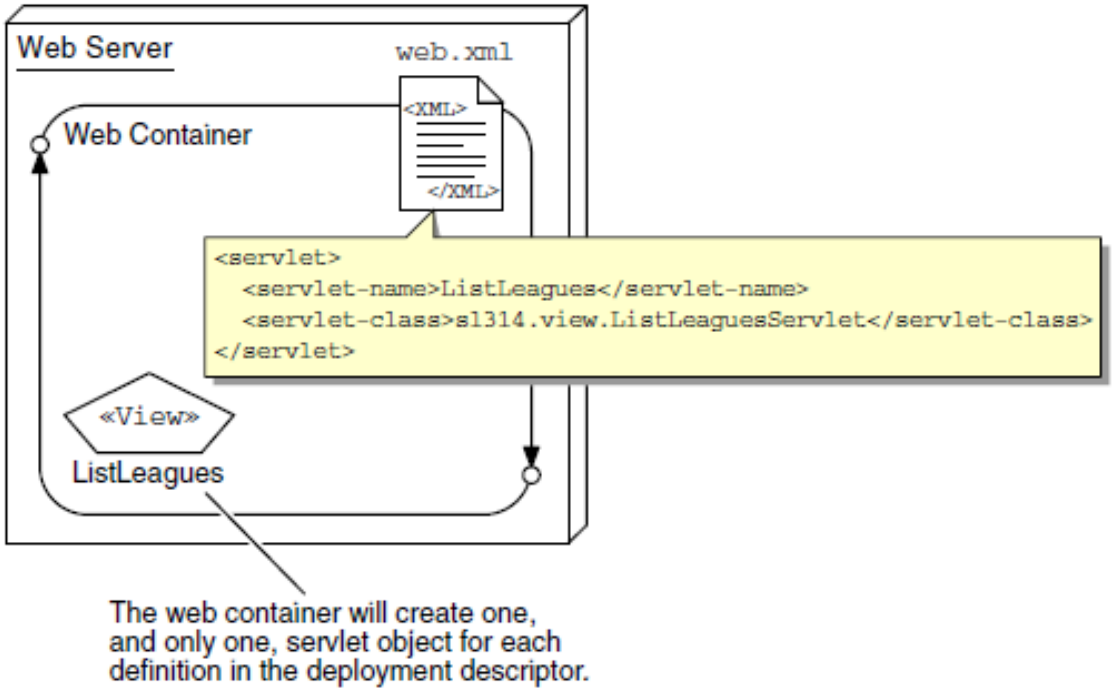
```
56      out.println("  <td><h3>" + pageTitle + "</h3></td>");
57      out.println("</tr>");
58      out.println("</table>");
59
60      // Generate main body
61      out.println("<p>");
62      out.println("The set of soccer leagues are:");
63      out.println("</p>");
64
65      out.println("<ul>");
66      Iterator items = leagueList.iterator();
67      while ( items.hasNext() ) {
68        League league = (League) items.next();
69        out.println("  <li>" + league.getTitle() + "</li>");
70      }
71      out.println("</ul>");
72
73      out.println("</body>");
74      out.println("</html>");
75    } // END of doGet method
```

Soccer League Web Application Structure

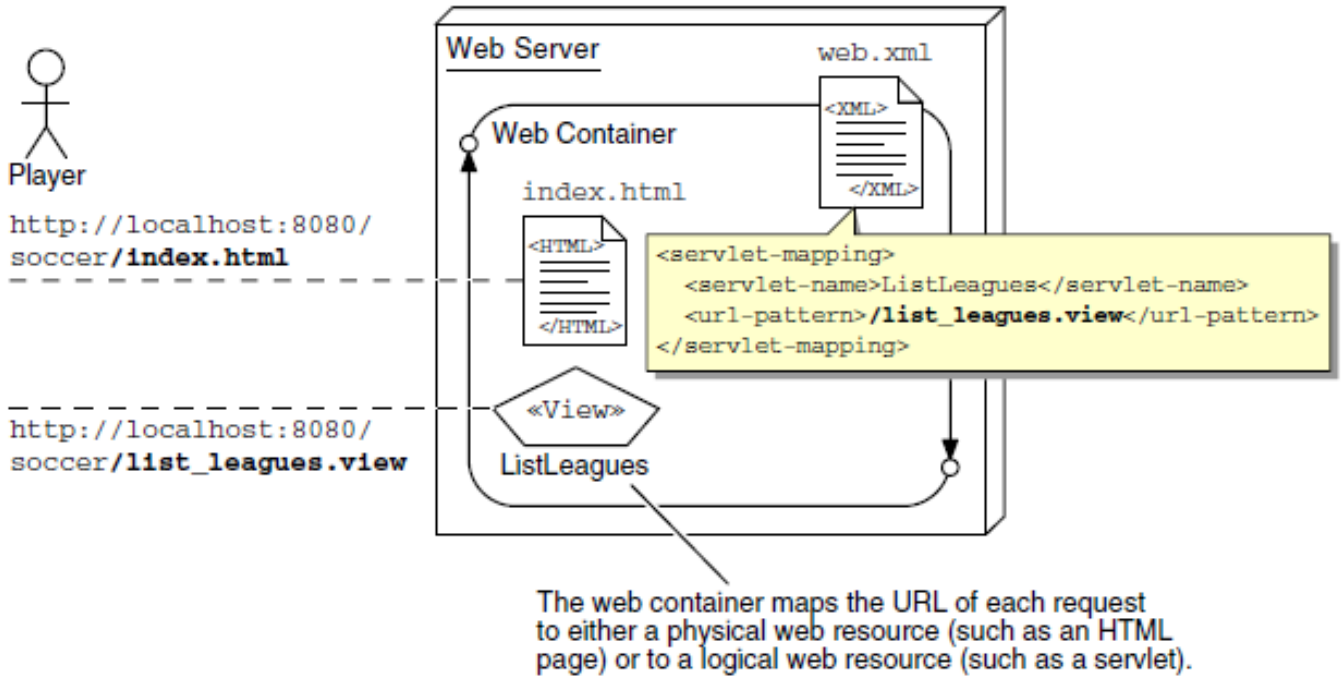
The logical web application hierarchy:

The physical web application hierarchy:

Configuring a Servlet Definition



Configuring a Servlet Mapping



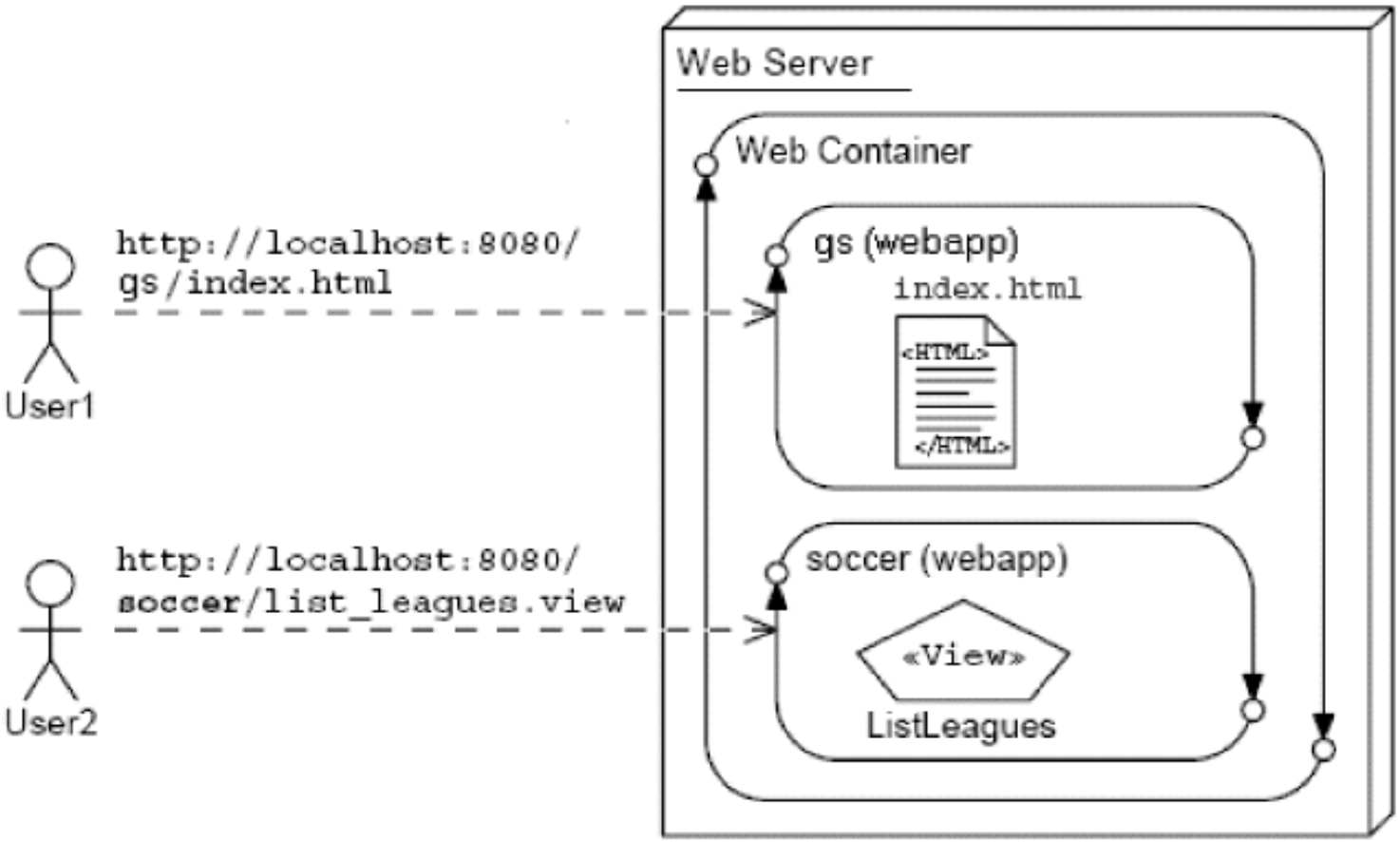
Complete Deployment Descriptor

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2
3   <web-app
4       xmlns="http://java.sun.com/xml/ns/j2ee"
5       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6       xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
7                           http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
8       version="2.4">
9
10    <display-name>SL-314 WebApp Example</display-name>
11    <description>
12      This Web Application demonstrates a single View servlet.
13    </description>
14
15    <servlet>
16      <servlet-name>ListLeagues</servlet-name>
17      <servlet-class>sl314.view.ListLeaguesServlet</servlet-class>
18    </servlet>
19
```

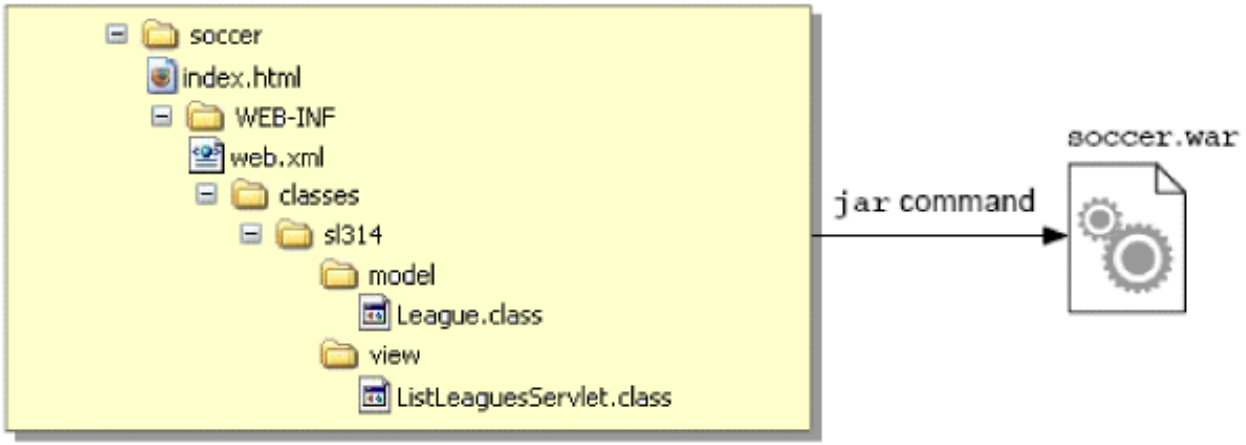
Complete Deployment Descriptor

```
1.<?xml version="1.0" encoding="UTF-8"?>
2.<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/
  javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="http://java.sun.com/xml/ns/
  javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
3.
4.   <servlet>
5.     <servlet-name>ListLeagues</servlet-name>
6.     <servlet-class>sl314.view.ListLeaguesServlet</servlet-
  class>
7.   </servlet>
8.
9.   <servlet-mapping>
10.     <servlet-name>ListLeagues</servlet-name>
11.     <url-pattern>/list_leagues.view</url-pattern>
12.   </servlet-mapping>
13.
```

Web Application Context Root



WAR Files for Deployment



Application Server deployment of a WAR file:

Activating the Servlet in a Web Browser

Request for `http://localhost:8080/soccer/index.html` presents:

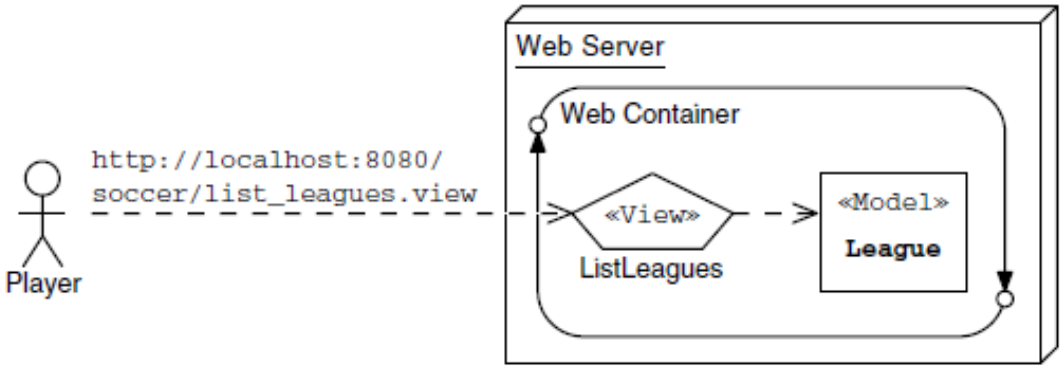
HTML:

```
20  <h3>Players</h3>
21
22  <ul>
23    <li><a href='list_leagues.view'>List all leagues</a></li>
24    <li>Register for a league (TBA)</li>
25  </ul>
```

Clicking on List performs a GET request for the URL:
`http://localhost:8080/soccer/list_leagues.view`

Activating the ListLeagues View

Request for the `list_league.view` is sent to the container:



This servlet generates this view:

The set of soccer leagues are:

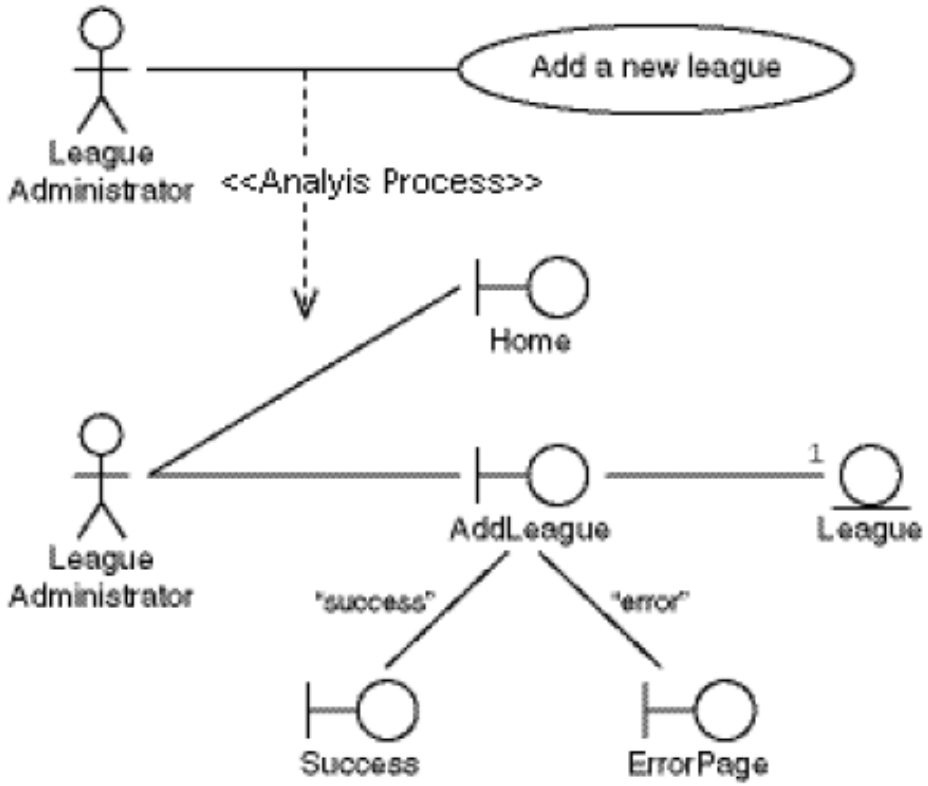
- Soccer League (Spring '03)
- Summer Soccer Fest 2003
- Fall Soccer League (2003)
- Soccer League (Spring '04)
- The Summer of Soccer Love 2004
- Fall Soccer League (2004)

Developing a controller component

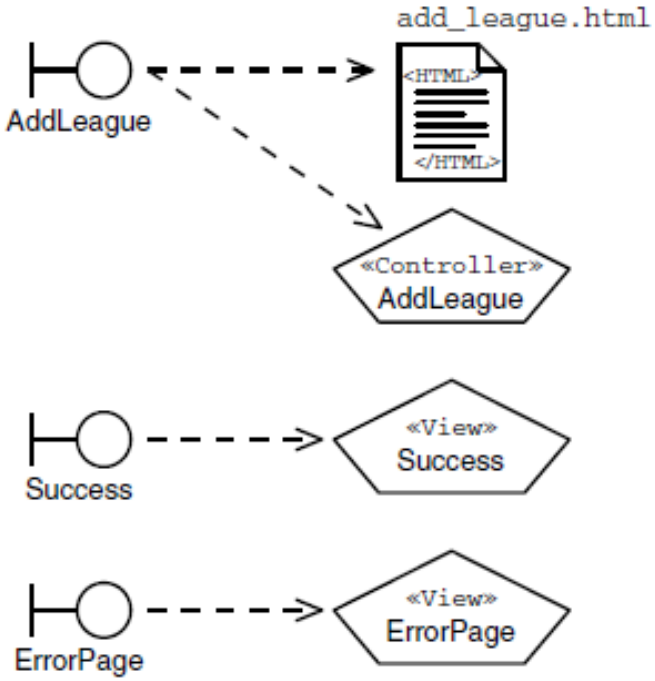
Types of Controller Components

- Process input from a user
- Support screen navigation
- Prepare data for view components

Add a New League Analysis Model

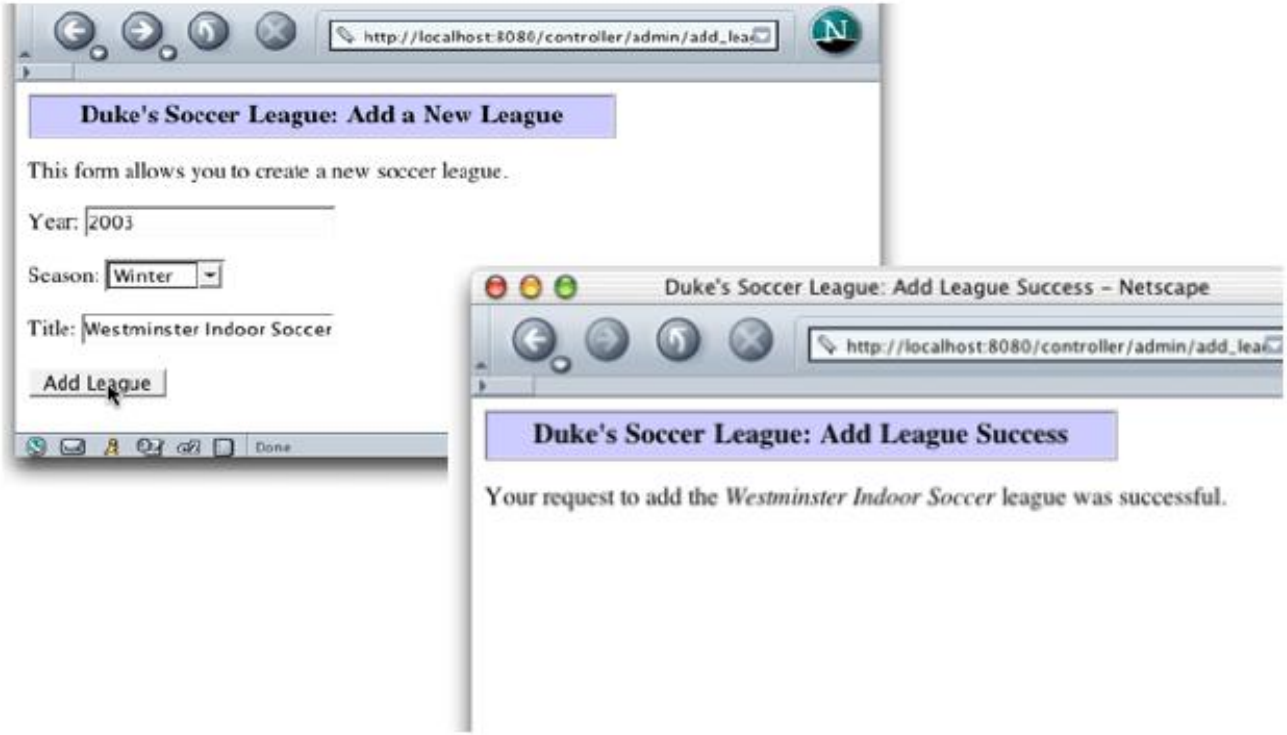


Add League Boundary Components



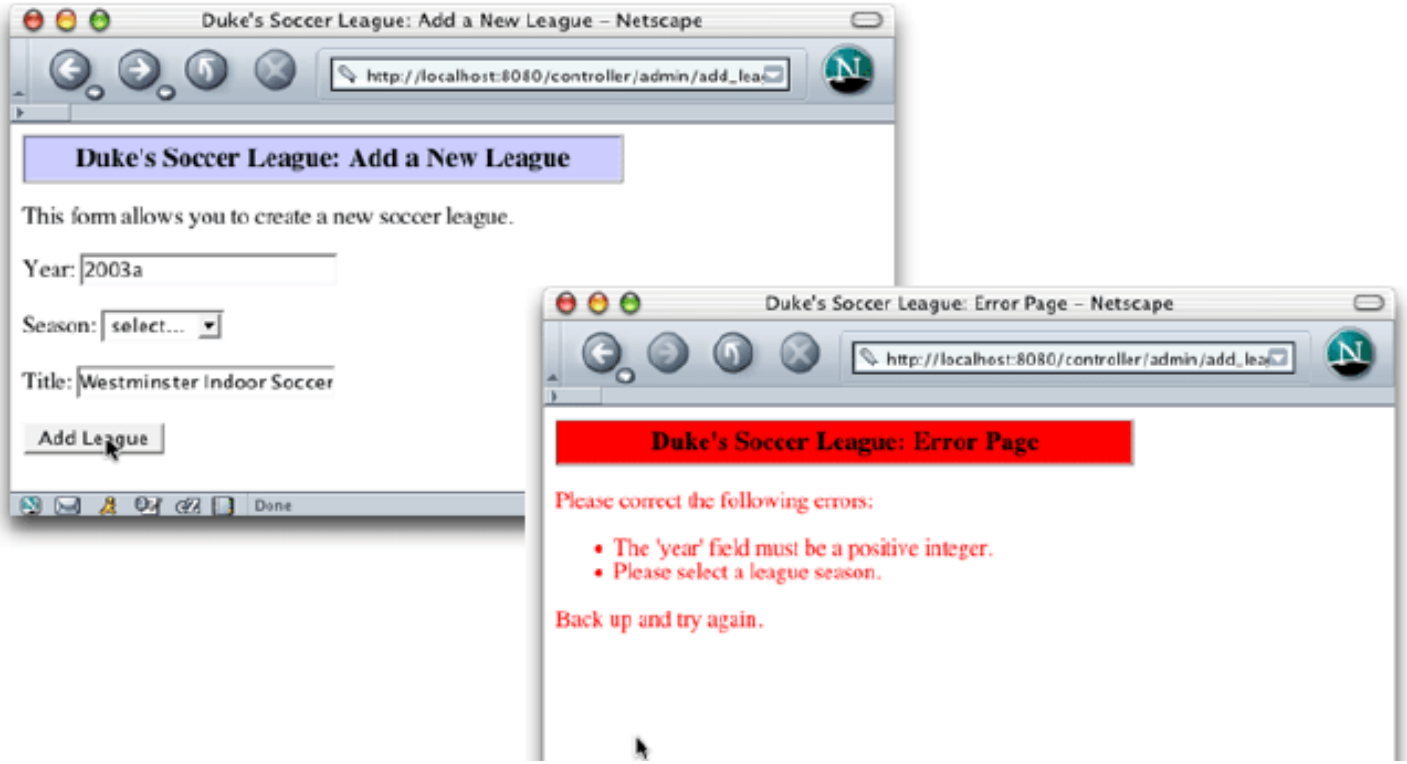
Add a New League Page Flow

Success path:

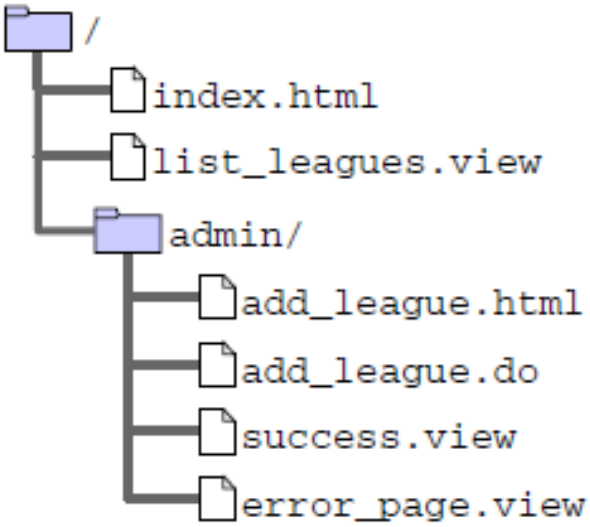


Add a New League Page Flow (continued)

Error path:



Soccer League Web Structure



Creating an HTML Form

The `form` Tag

The following is a partial structure of an HTML form:

```
<form action='URL TO CONTROLLER' method='GET or POST'>
<!-- PUT FORM COMPONENT TAGS HERE -->
</form>
```

For example:

```
<form action='add_league.do' method='POST'>
Year: [textfield tag]
Season: [drop-down list tag]
Title: [textfield tag]
[submit button tag]
</form>
```

A single web page can contain many forms.

Textfield Component

In Netscape™, a textfield component looks like this:

This form allows you to create a new soccer league.

Year: 2003

The HTML content for this component is:

```
16  <p>
17  This form allows you to create a new soccer league.
18  </p>
19
20  <form action='add_league.do' method='POST'>
21  Year: <input type='text' name='year' /> <br/><br/>
```

Drop-Down List Component

In Netscape, a drop-down list component looks like this:

The HTML content for this component is:

```
22  Season: <select name='season'>
23            <option value='UNKNOWN'>select...</option>
24            <option value='Spring'>Spring</option>
25            <option value='Summer'>Summer</option>
26            <option value='Fall'>Fall</option>
27            <option value='Winter'>Winter</option>
28          </select> <br/><br/>
```

Submit Button

In Netscape, a submit button component might look like this

The HTML content for this component is:

```
29  Title: <input type='text' name='title' /> <br/><br/>
30  <input type='submit' value='Add League' />
31  </form>
```

Complete Add a New League Form

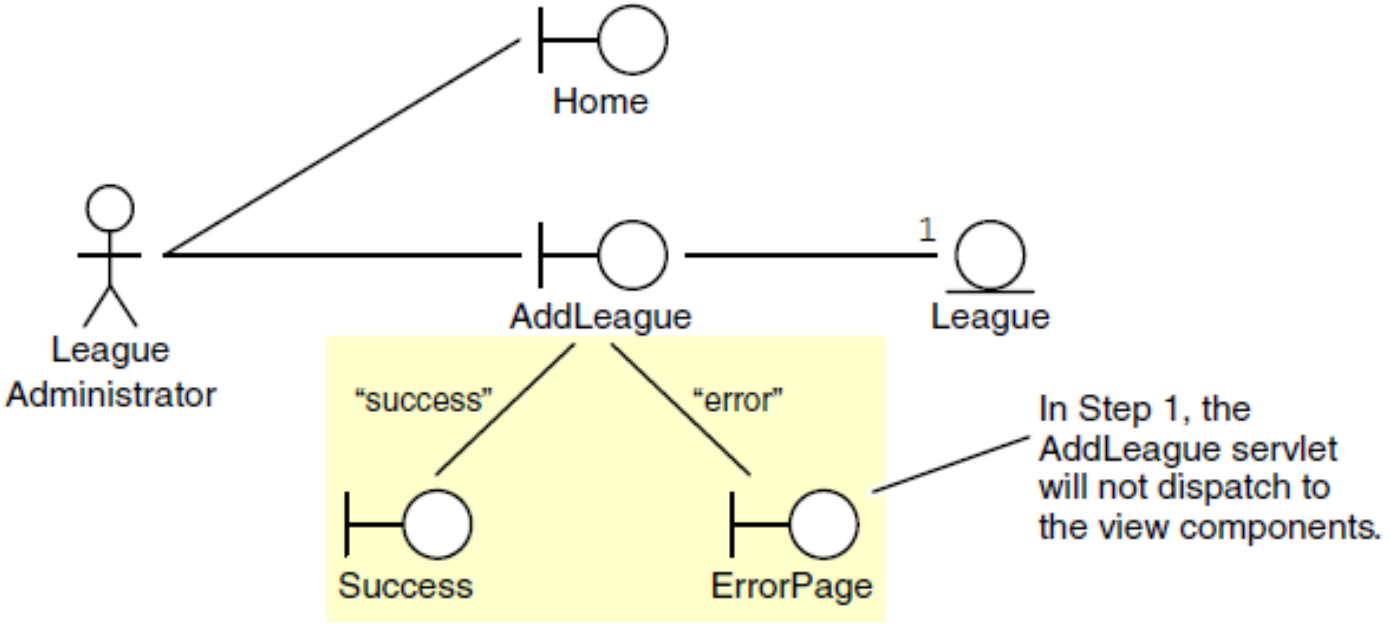
```
16  <p>
17  This form allows you to create a new soccer league.
18  </p>
19
20  <form action='add_league.do' method='POST'>
21  Year: <input type='text' name='year' /> <br/><br/>
22  Season: <select name='season'>
23            <option value='UNKNOWN'>select...</option>
24            <option value='Spring'>Spring</option>
25            <option value='Summer'>Summer</option>
26            <option value='Fall'>Fall</option>
27            <option value='Winter'>Winter</option>
28          </select> <br/><br/>
29  Title: <input type='text' name='title' /> <br/><br/>
30  <input type='submit' value='Add League' />
31  </form>
```

Developing a Controller Servlet

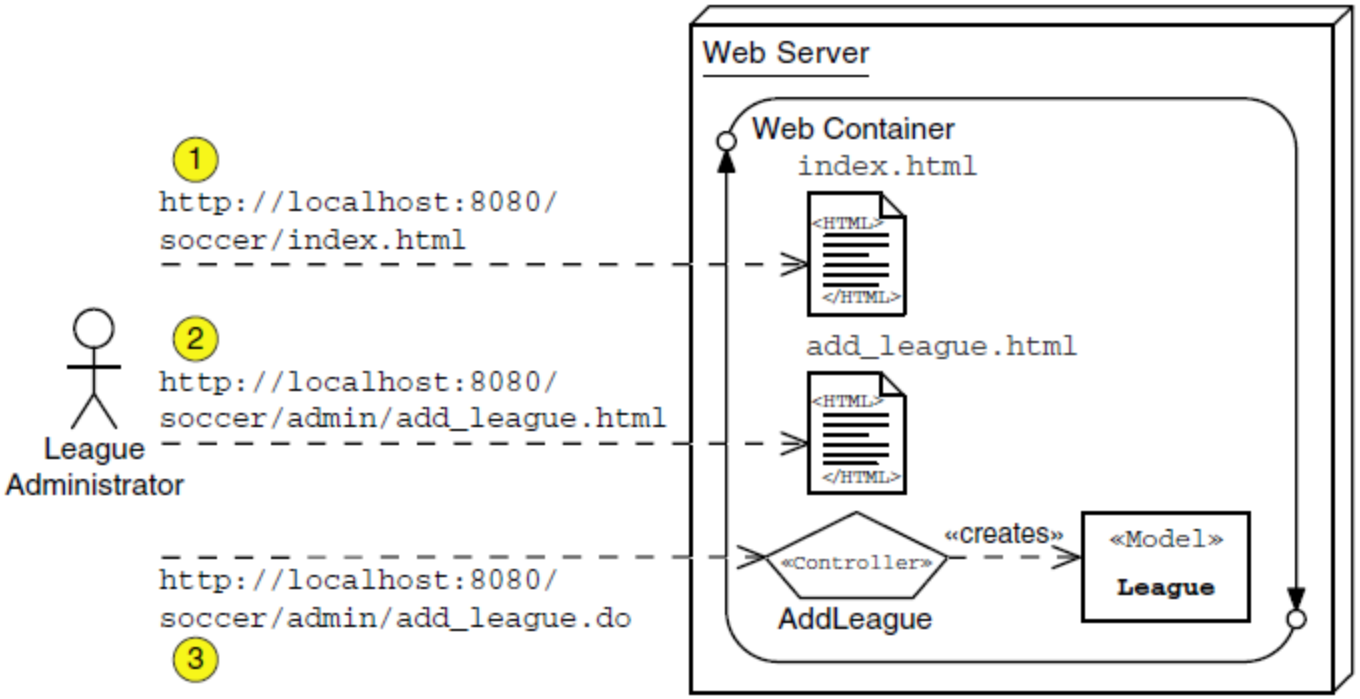
A form-processing (controller) servlet needs to:

1. Retrieve form parameters from the HTTP request.
2. Perform any data conversion on the form parameters.
3. Verify the form parameters.
4. Execute the business logic.
5. Dispatch to the next view component based on the results of the previous steps.

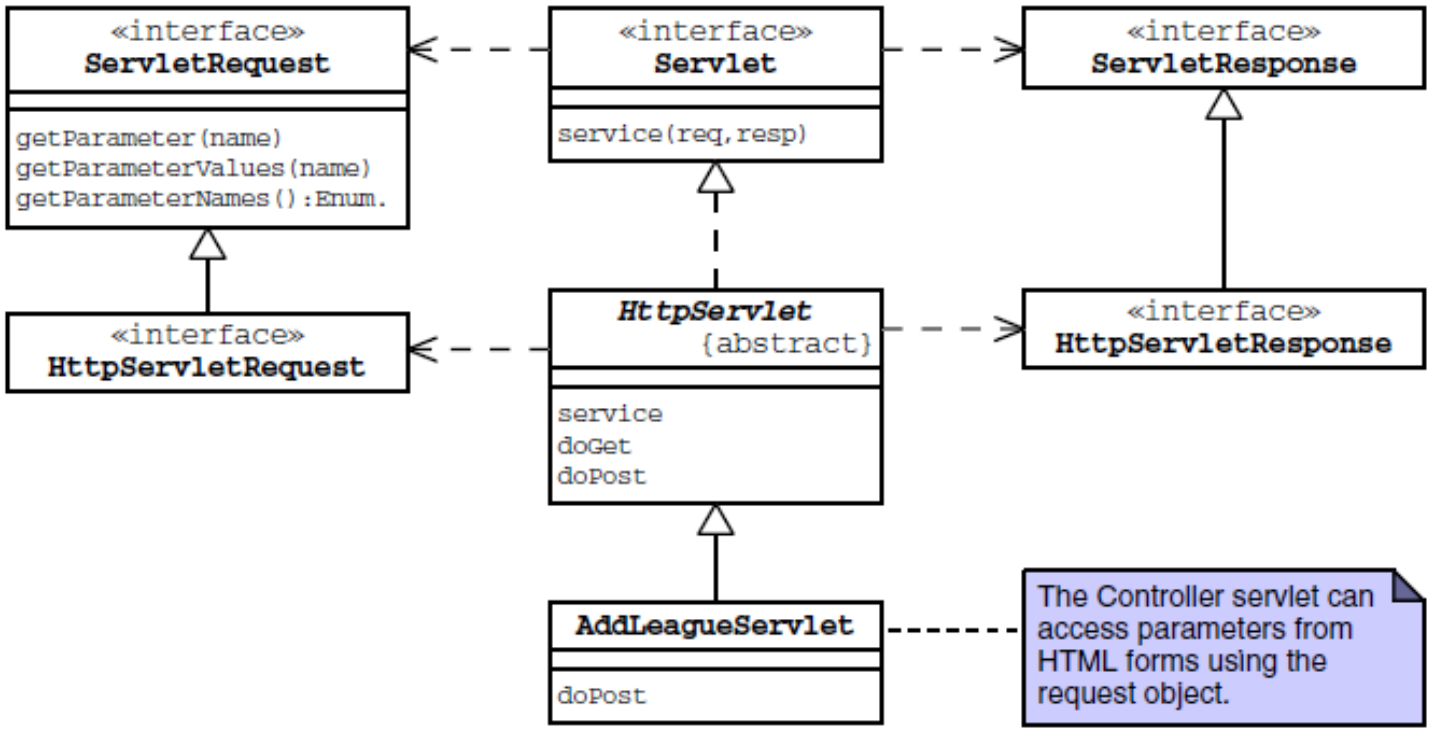
Add League Analysis Model (Stage 1)



Add League Architecture Model (Stage 1)(continued)



Servlet API to Retrieve Form Parameters



The AddLeagueServlet Class Declaration

```
1   package sl314.controller;
2
3   import javax.servlet.http.HttpServlet;
4   import javax.servlet.http.HttpServletRequest;
5   import javax.servlet.http.HttpServletResponse;
6   import javax.servlet.ServletException;
7   // Support classes
8   import java.io.IOException;
9   import java.io.PrintWriter;
10  // Model classes
11  import sl314.model.League;
12  import java.util.List;
13  import java.util.LinkedList;
14
15  public class AddLeagueServlet extends HttpServlet {
16    public void doPost(HttpServletRequest request,
17                       HttpServletResponse response)
18          throws IOException, ServletException {
19
20      // Keep a set of strings to record form processing errors.
21      List errorMsgs = new LinkedList();
```

Retrieving Form Parameters and Data Conversion

```
22
23      try {
24
25        // Retrieve form parameters.
26        String yearStr = request.getParameter("year").trim();
27        String season = request.getParameter("season").trim();
28        String title = request.getParameter("title").trim();
29
30        // Perform data conversions.
31        int year = -1;
32        try {
33          year = Integer.parseInt(yearStr);
34        } catch (NumberFormatException nfe) {
35         errorMsgs.add("The 'year' field must be a positive integer.");
36        }
37
```

Performing Form Validations

```
37
38        // Verify form parameters
39        if ( (year != -1) && ((year < 2000) || (year > 2010)) ) {
40          errorMsgs.add("The 'year' field must within 2000 to 2010.");
41        }
42        if ( season.equals("UNKNOWN") ) {
43          errorMsgs.add("Please select a league season.");
44        }
45        if ( title.length() == 0 ) {
46          errorMsgs.add("Please enter the title of the league.");
47        }
48
49        // Send the ErrorPage view if there were errors
50        if ( ! errorMsgs.isEmpty() ) {
51          // dispatch to the ErrorPage
52          PrintWriter out = response.getWriter();
53          out.println("ERROR PAGE");
54          return;
55        }
56
```

Performing the Business Logic

```
57
58        // Perform business logic
59        League league = new League(year, season, title);
60
61        // Send the Success view
62        PrintWriter out = response.getWriter();
63        out.println("SUCCESS");
64        return;
65
```

Handling an Exception

```
65
66      // Handle any unexpected exceptions
67      } catch (RuntimeException e) {
68        errorMsgs.add(e.getMessage());
69        // dispatch to the ErrorPage
70        PrintWriter out = response.getWriter();
71        out.println("ERROR PAGE");
72
73        // Log stack trace
74        e.printStackTrace(System.err);
75
76      } // END of try-catch block
77    } // END of doPost method
78  } // END of AddLeagueServlet class
```

Developing the AddLeagueServlet Code

```
6   import javax.servlet.RequestDispatcher;
7   import javax.servlet.ServletException;
8   // Support classes
9   import java.io.IOException;
10  import java.io.PrintWriter;
11  // Model classes
12  import sl314.model.League;
13  import java.util.List;
14  import java.util.LinkedList;
15
16  public class AddLeagueServlet extends HttpServlet {
17    public void doPost(HttpServletRequest request,
18                       HttpServletResponse response)
19          throws IOException, ServletException {
20
21      // Keep a set of strings to record form processing errors.
22      List errorMsgs = new LinkedList();
23      // Store this set in the request scope, in case we need to
24      // send the ErrorPage view.
25      request.setAttribute("errorMsgs", errorMsgs);
26
```

Developing the `AddLeagueServlet` Code (Part 2)

```
27      try {
28
29        // Retrieve form parameters.
30        String yearStr = request.getParameter("year").trim();
31        String season = request.getParameter("season").trim();
32        String title = request.getParameter("title").trim();
33
34        // Perform data conversions.
35        int year = -1;
36        try {
37          year = Integer.parseInt(yearStr);
38        } catch (NumberFormatException nfe) {
39          errorMsgs.add("The 'year' field must be a positive integer.")
40        }
41
```

Developing the AddLeagueServlet Code (Part 3)

```
41
42        // Verify form parameters
43        if ( (year != -1) && ((year < 2000) || (year > 2010)) ) {
44          errorMsgs.add("The 'year' field must within 2000 to 2010.");
45        }
46        if ( season.equals("UNKNOWN") ) {
47          errorMsgs.add("Please select a league season.");
48        }
49        if ( title.length() == 0 ) {
50          errorMsgs.add("Please enter the title of the league.");
51        }
52
53        // Send the ErrorPage view if there were errors
54        if ( ! errorMsgs.isEmpty() ) {
55          RequestDispatcher view
56            = request.getRequestDispatcher("error_page.view");
57          view.forward(request, response);
58          return;
59        }
60
```

Developing the AddLeagueServlet Code (Part 4)

```
61        // Perform business logic
62        League league = new League(year, season, title);
63        // Store the new league in the request-scope
64        request.setAttribute("league", league);
65
66        // Send the Success view
67        RequestDispatcher view
68          = request.getRequestDispatcher("success.view");
69        view.forward(request, response);
70        return;
71
72      // Handle any unexpected exceptions
73      } catch (RuntimeException e) {
74        errorMsgs.add(e.getMessage());
75        RequestDispatcher view
76          = request.getRequestDispatcher("error_page.view");
77        view.forward(request, response);
78
79        // Log stack trace
80        e.printStackTrace(System.err);
```

The SuccessServlet Code

```
11
12  public class SuccessServlet extends HttpServlet {
13
14    public void doGet(HttpServletRequest request,
15                      HttpServletResponse response)
16          throws IOException {
17      generateView(request, response);
18    }
19
20    public void doPost(HttpServletRequest request,
21                       HttpServletResponse response)
22          throws IOException {
23      generateView(request, response);
24    }
25
26    public void generateView(HttpServletRequest request,
27                             HttpServletResponse response)
28          throws IOException {
29
```

The SuccessServlet Code (Part 2)

```
30      // Set page title
31      String pageTitle = "Duke's Soccer League: Add League Success";
32
33      // Retrieve the 'league' from the request-scope
34      League league = (League) request.getAttribute("league");
35
36      // Specify the content type is HTML
37      response.setContentType("text/html");
38      PrintWriter out = response.getWriter();
39
40      // Generate the HTML response
41      out.println("<html>");
```

The SuccessServlet Code (Part 3)

```
54
55      // Generate main body
56      out.println("<p>");
57      out.print("Your request to add the ");
58      out.print("<i>" + league.getTitle() + "</i>");
59      out.println(" league was successful.");
60      out.println("</p>");
61
62      out.println("</body>");
63      out.println("</html>");
64
65    } // END of generateResponse method
66
67  } // END of SuccessServlet class
```