$01KPS_{BF}$
Progettazione di applicazioni web

Introduction to Java Server Pages

Fulvio Corno, Alessio Bosca

Dipartimento di Automatica e Informatica

Politecnico di Torino

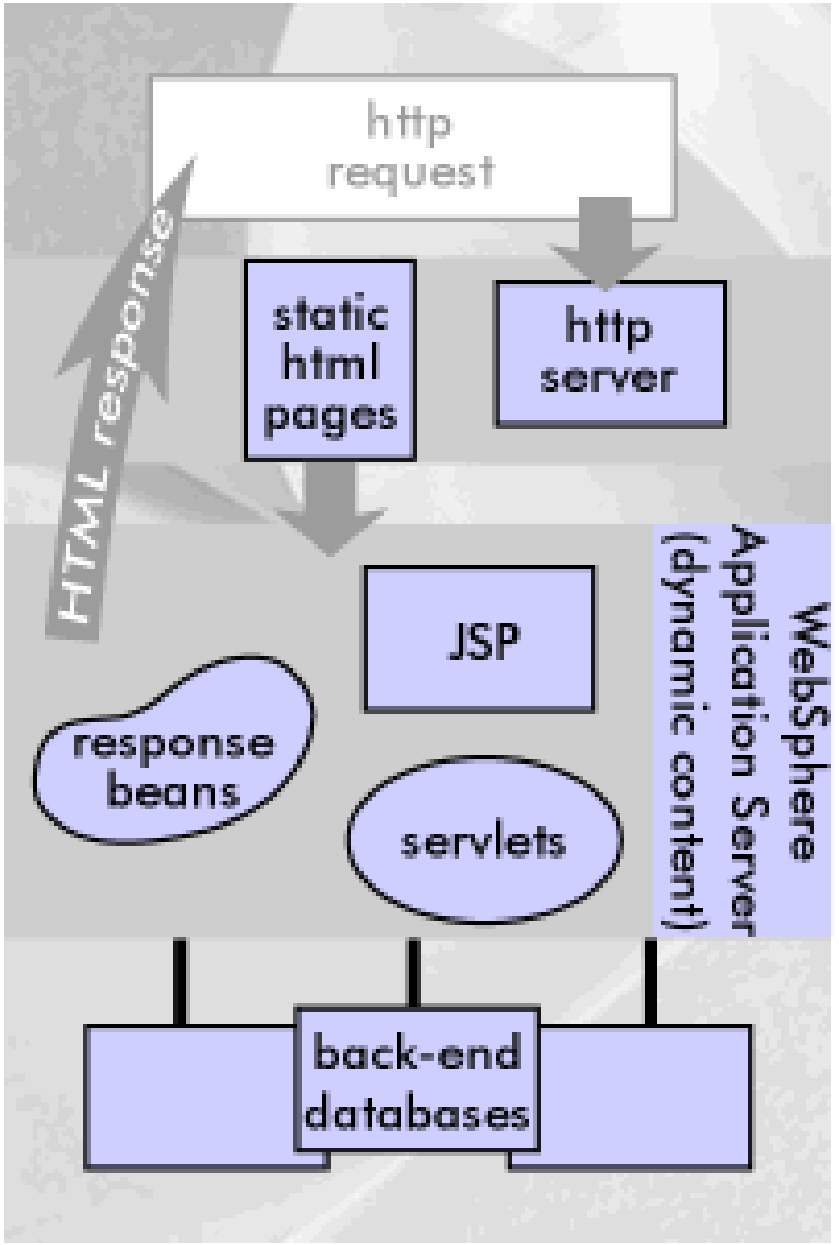
Introduction to Java Server Pages

Part I

Basic Java Server Pages

Presentation Overview

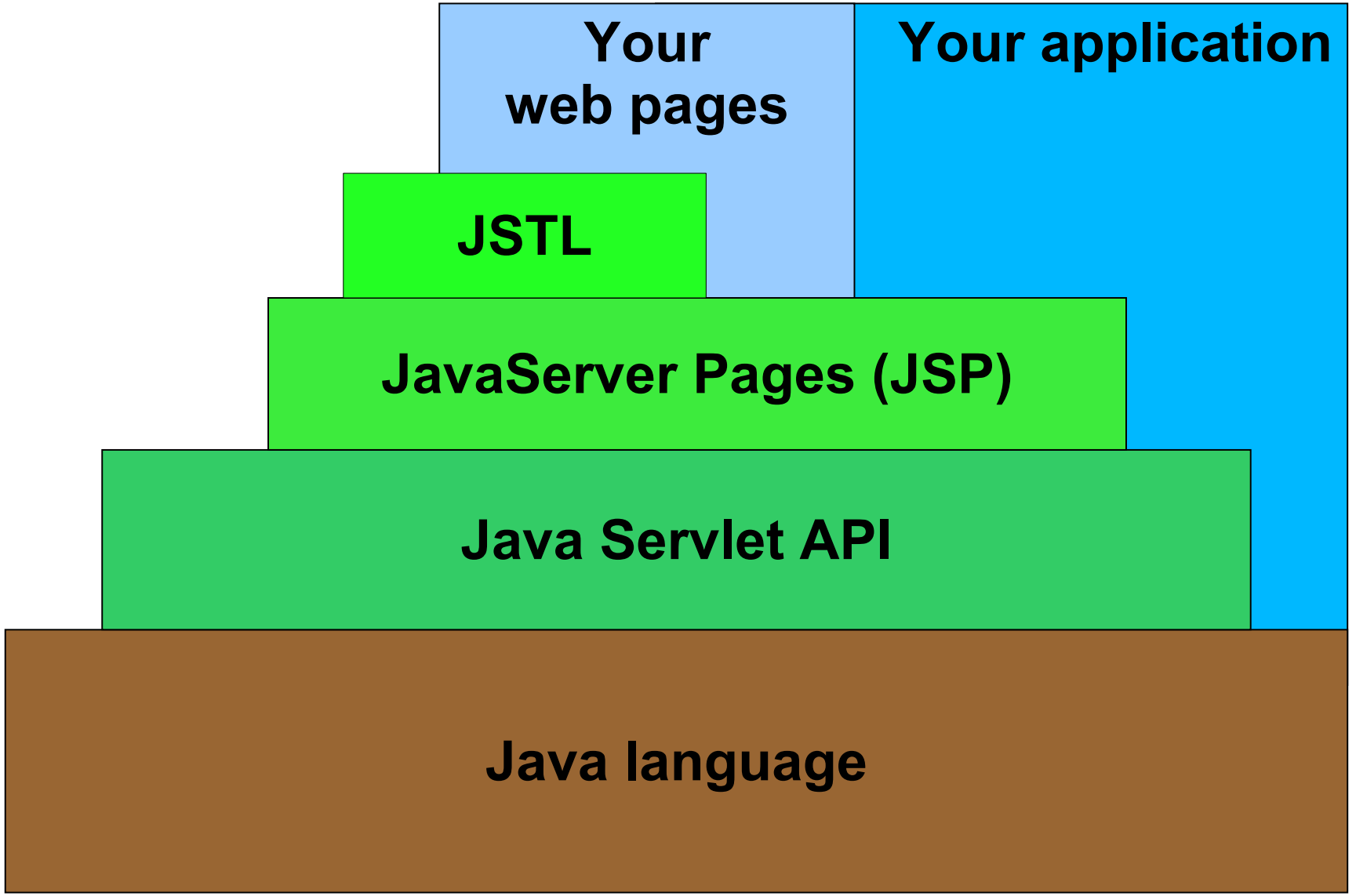
- What are Java Server Pages?
- Structure of a JSP document.
  - Scriplet Tag
  - Expression Tag
  - Declaration Tag
  - Directive Tag
  - JSP Tags
- Processing Request Parameters in JSPs.



The J2EE presentation tier

- Servlets
  - Java classes that handle requests by producing responses (e.g., HTTP requests and responses)
- JavaServer Pages (JSP)
  - HTML-like pages with some dynamic content.
  - Translated into servlets automatically
- JSP Standard Tag Library (JSTL)
  - Set of standard components for JSP
  - Used inside JSP pages.

Organization of the platform



Why use JSP Technology?

- Convenient:
  - We already know Java and HTML!
- Provides an extensive infrastructure for:
  - Tracking sessions.
  - Managing cookies.
  - Reading and sending HTML headers.
  - Parsing and decoding HTML form data.
- Efficient:
  - Every request for a JSP is handled by a simple Java thread.

Why use JSP Technology?

- Portable
  - JSP follow a well standardized API.
  - The Java VM which is used to execute a JSP file is supported on many architectures and operating systems.
- Inexpensive
  - There are a number of free or inexpensive Web Servers
  - that are good for commercial-quality websites.

What is JSP?

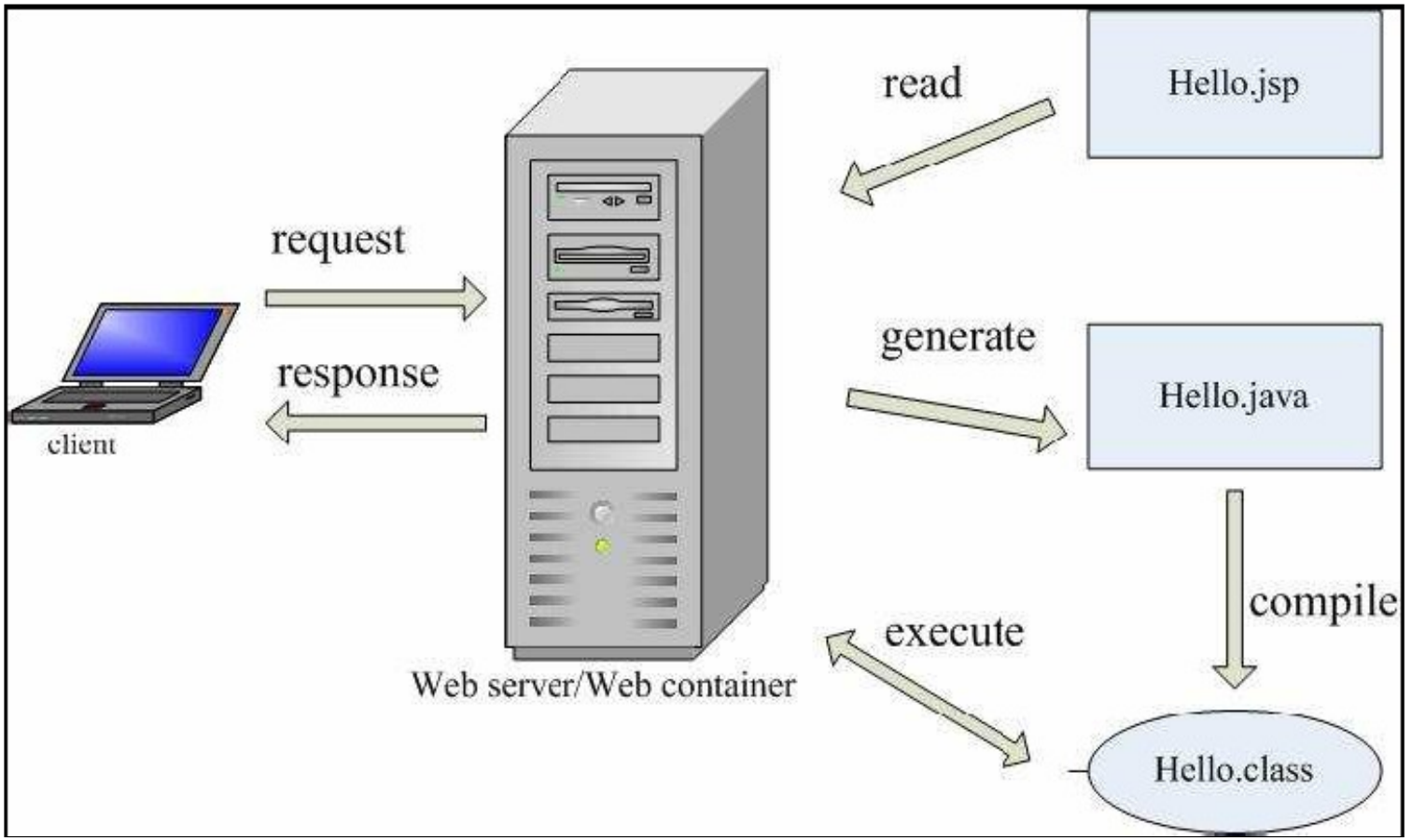
- Java based technology that simplifies the developing of dynamic web sites
- JSP pages are HTML pages with embedded code that allows to access data from Java code running on the server
- JSP provides separation of HTML presentation logic from the application logic.

JSP Technology

- JSP technology provides a way to combine the worlds of HTML and Java servlet programming.
- JSP specs are built on the Java Servlet API.
- JSP supports two different styles for adding dynamic content to web pages:
  - JSP pages can embed actual programming code (typically Java)
  - JSP supports a set of HTML-like tags that interact with Java objects on the server (without the need for raw Java code to appear in the page)

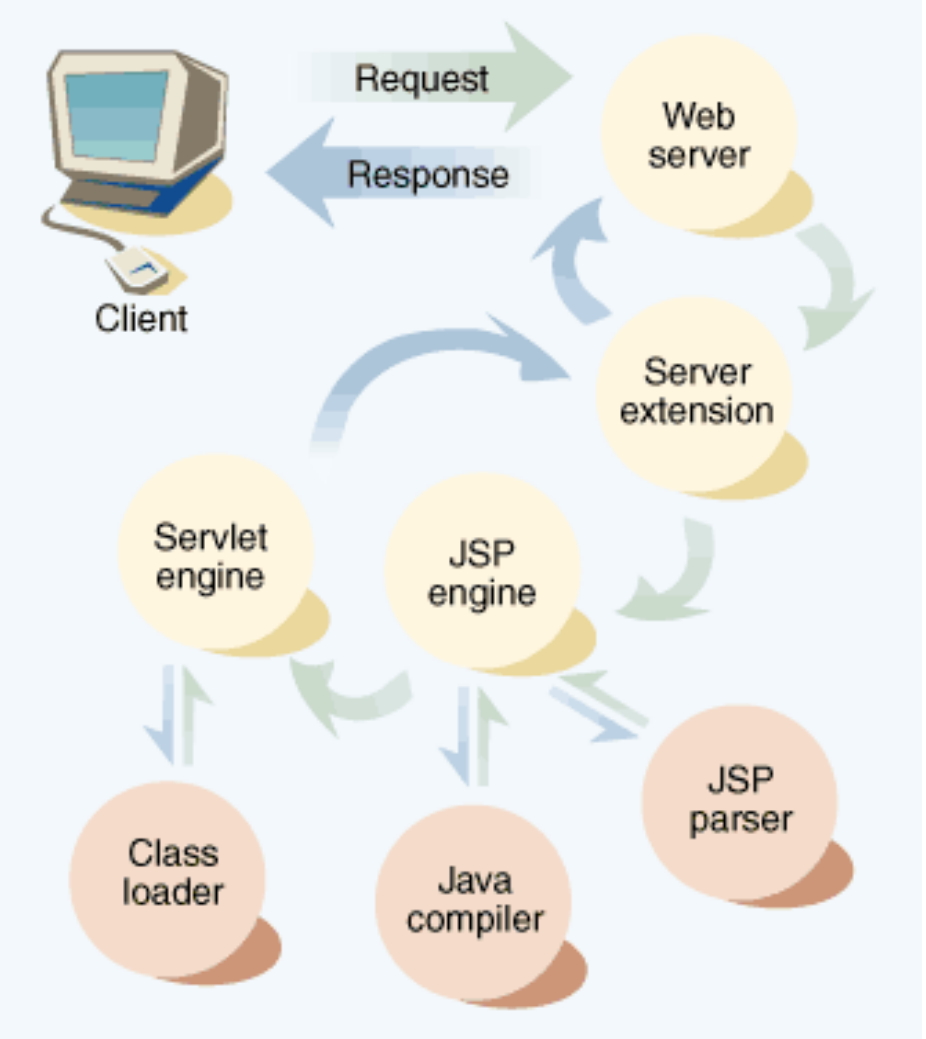
JSP Flow..

- JSP pages “live” within a container that manages its interaction:
  - HTTP Protocol (request, response, header)
  - Sessions



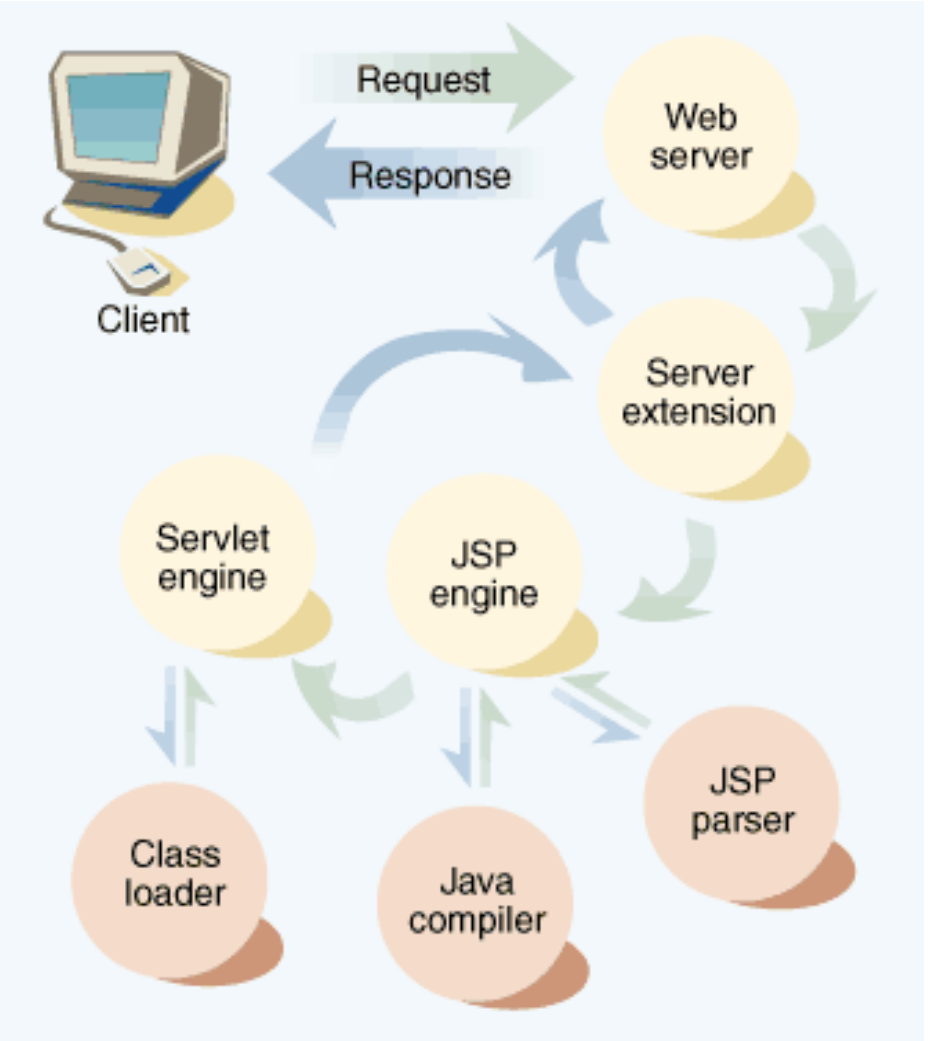
How it really works... (1/2)

- Client requests a page ending with “.jsp”
- Web Server fires up the JSP engine
- JSP engine checks whether JSP file is new or changed
- JSP engine converts the page into a Java servlet (JSP parser)
- JSP engine compiles the servlet (Java compiler)



How it really works... (2/2)

- Servlet Engine executes the new Java servlet using the standard API
- Servlet's output is transferred by Web Server as a http response



Structure of a JSP file.

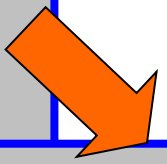
- Similar to a HTML document. Four basic tags:
  - Scriplet
  - Expression
  - Declaration
  - Definition

```
<html>
  <head>
    <title>Hello World</head>
  </head>
<body>
<h1>Hello, World</h1>
It's <%=(new java.util.Date()).toString()%>
and all is well.
</body>
</html>
```

Scriptlet Tag

- Two forms:
  - `<% any java code %>`
  - `<jsp:scriptlet> ... </jsp:scriptlet>`
    - (XML form)
- Embeds Java code in the JSP document that will be executed each time the JSP page is processed.
- Code is inserted in the service() method of the generated Servlet

```
<html>
<body>
    <% for (int i = 0; i < 2; i++) { %>
        <p>Hello World!</p>
    <% } %>
</body>
</html>
```

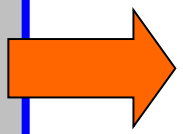


```
<html>
<body>
    <p>Hello World!</p>
    <p>Hello World!</p>
</body>
</html>
```

Expression Tag

- <%= *expr* %>
- <jsp:expression> *expr* </jsp:expression>
- Expression *expr* is evaluated (in Java) and its value is placed in the output.
  - Note: no semi-colon ";" following `expr`

```
<html>
<body>
<p>
<%= Integer.toString( 5 * 5 ) %>
</p>
</body>
</html>
```



```
html>
<body>
        <p>25</p>
</body>
</html>
```

(Embedded) Expression language

- An EL expression always starts with a ${ and ends with a }
- All EL expressions are evaluated at **runtime**
- The EL usually handles data type conversion and null values -> easy to use
- The expression can include
  - literals (“1”, “100” etc)
  - variables
  - implicit variables

Examples

- `${1+2+3}`
- `${param.Address}`

EL Operators

```
==    !=    <     >     <=    >=
+     /     div   -     *
&&    and   ||    or    !     not
empty
```

JSP Implicit Objects

- In JSP, need to be able to access information about the environment in which the page is running e.g. the **parameters** passed in a request for a form, the **browser type** of the user etc.
- Implicit objects are a set of Java objects that the JSP Container makes available to developers **in each page**. These objects may be accessed as built-in variables via scripting elements
- The JSTL EL allows these objects to be accessed as '**Implicit Variables**'
- Implicit variable are just pre-agreed fixed variable names that can be used in JSTL Expressions
  - Think of as "variables that are automatically available to your JSP page"

Implicit Objects in Expression language

- Very common implicit object is `param`
  - `param` refers to parameter passed in a request message (e.g. information entered into a form by a user).
- Example
  - `${param.userName}`

Declaration Tag

- <%! *declaration* %>
- <jsp:declaration>
  declaration(s)</jsp:declaration>
- Embeds Java declarations inside a JSP document
- Code is inserted in the body of the servlet class, **outside the service method**.
  - May declare instance variables
  - May declare (private) member functions

```
<html>
<body>
<%! private int accessCount = 0; %>
   <p> Accesses to page since server reboot:
   <%= ++accessCount %> </p>
</body>
</html>
```

Warning!

- JSP declarations add variables in the servlet instance class
  - Variables shared by all threads (all requests to the same servlet)
  - Until servlet container unloads servlet
  - Beware simultaneous access! Must use synchronized methods

```
<html>
<body>
<%! private int accessCount = 0; %>
    <p> Accesses to page since server reboot:
    <%= ++accessCount %> </p>
</body>
</html>
```

Directive Tag

- `<%@ directive att="value" %>`
- `<jsp:directive.page att="val" />`
- Directives are used to convey special processing information about the page to the JSP container.
  - `page` directive
  - `include` directive

```
<%@ page import="java.util.*" %>
<%@ page contentType="text/xml" %>
<%@ page errorPage="error.jsp" %>
```

The JSP @page Directive

- `import="package.class"` or `import="pkg.class1,...,pkg.classN"`
  - This lets you specify what packages should be imported. The import attribute is the only one that is allowed to appear multiple times.
  - Example: <%@ page import="java.util.*" %>
- `contentType="MIME-Type"` or `contentType="MIME-Type; charset=Character-Set"`
  - Specifies the MIME type of the output. Default is `text/html`.
  - Example: <%@ page contentType="text/plain" %> equivalent to <% response.setContentType("text/plain"); %>

The JSP @page Directive

- `session="true|false"`
  - A value of **true** (the default) indicates that the predefined variable session (of type HttpSession) should be bound to the existing session if one exists, otherwise a new session should be created and bound to it.
  - A value of false indicates that no sessions will be used, and attempts to access the variable session will result in errors at the time the JSP page is translated into a servlet.

The JSP @page Directive

- `errorPage="url"`
  - This specifies a JSP page that should process any Throwables thrown but not caught in the current page.
- `isErrorPage="true|false"`
  - This indicates whether or not the current page can act as the error page for another JSP page. The default is false.

The JSP @page Directive

- `isThreadSafe="true|false"`
  - A value of true (the default) indicates normal servlet processing, where multiple requests can be processed simultaneously with a single servlet instance, under the assumption that the author synchronized access to instance variables.
  - A value of false indicates that the servlet should implement SingleThreadModel, with requests either delivered serially or with simultaneous requests being given separate servlet instances.
    - Don’t use it, since it reduces performance!

The JSP @page Directive

- `buffer="sizekb|none"`
  - This specifies the buffer size for the jspWriter out. The default is server-specific, but must be at least 8kb.
- `autoflush="true|false"`
  - A value of true, the default, indicates that the buffer should be flushed when it is full.
  - A value of false, rarely used, indicates that an exception should be thrown when the buffer overflows.
  - A value of false is illegal when also using buffer="none".

The JSP @page Directive

- `extends="package.class"`
  - This indicates the superclass of servlet that will be generated. Use this with extreme caution, since the server may be using a custom superclass already.
- `info="message"`
  - This defines a string that can be retrieved via the getServletInfo method.
- `language="java"`
  - Java is both the default and the only legal choice.

The JSP @include Directive

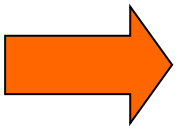
- `<%@ include file="`*`relative url`*`" %>`
  - Include files at the time the JSP page is translated into a servlet.
- The contents of the included file are parsed as regular JSP text, and thus can include static HTML, scripting elements,directives, and actions.
- Warning: when included files change, the page is **not** automatically recompiled

```
<%@ include file="header.jsp" %>
Only the content of a page is unique.
Header and footer are reused from header.jsp and footer.jsp
<%@ include file="footer.jsp" %>
```

JSP Comments

- Regular (HTML) Comment
  - `<!-- comment -->`
- Hidden (JSP) Comment
  - `<%-- comment --%>`

```
<html>
<!-- Regular Comment -->
<%-- Hidden Comment --%>
<%
  // Java comment
%>
</html>
```



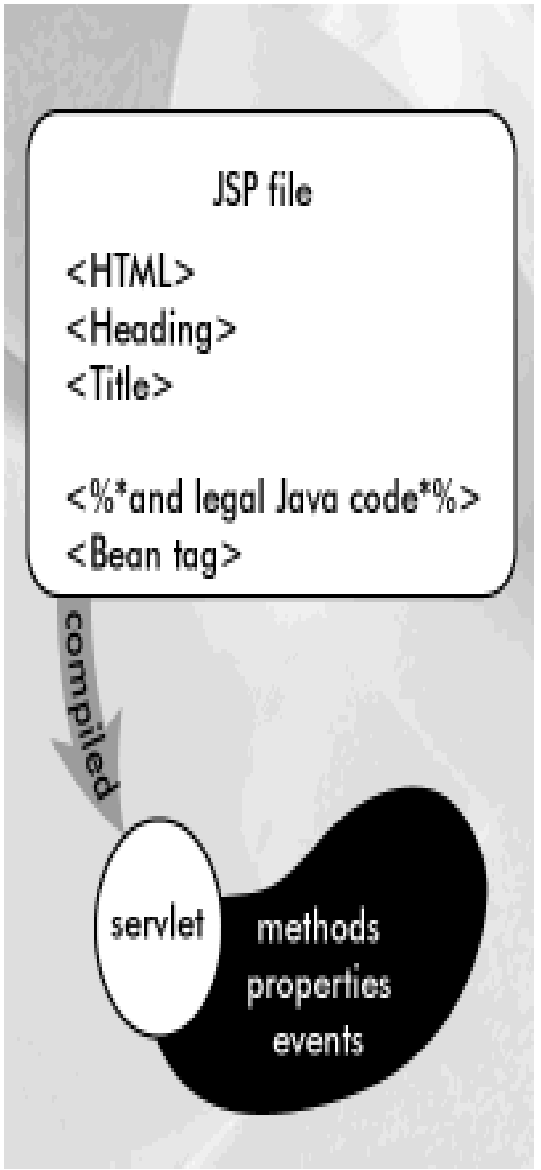
```
<html>
<!-- Regular Comment -->
</html>
```

Scriptlet Example

```
<html>
<head>
    <title>Scriptlet Example</title>
<%! public long fact (long x) {
        if (x == 0) return 1;
        else return x * fact(x-1);
    } %>
</head>

<body>

<table>
<tr> <th width="50"> x</th><th width="50">x! </th></tr>
<br>
<% for (int x = 1; x < 10; x++) { %>
    <tr><td width="50"><%= x%> </td><td width="50"> <%= fact(x) %></td></tr>
<% } %>
</table>

</body>
</html>
```

JSP Pages content

- Actions
  - `<% Any Java code... %>`
  - Goes into the service() method
- *Implicit objects* accessible to actions
  - `page`
  - `out`
  - `config`
  - `session`
  - `request`
  - `application`
  - `response`
  - `pageContext`
  - `exception`



Implicit Objects

- `request`
  - The HttpServletRequest parameter
  - Same usage as in servlets
  - Mainly used for **getting request parameters**
- `response`
  - The HttpServletResponse parameter
  - Same usage as in servlets
  - Rarely used in JSP (directives already to the work for us...)
- `out`
  - The **PrintWriter** associated to the **response** (buffered)
  - `out.println()`
  - Not much used... just escape to HTML
    - `%>html code<%`

Request object- getting parameters

- `String getParameter(String name)`
  - Returns the value of a request parameter as a String, or null if the parameter does not exist.
- `Map getParameterMap()`
  - Returns a java.util.Map of the parameters
- `Enumeration getParameterNames()`
  - Returns an Enumeration of String objects containing the names of the parameters
- `String[] getParameterValues(String name)`
  - Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
- More methods: http://java.sun.com/javaee/5/docs/api/
  - HttpServletRequest, ServletRequest

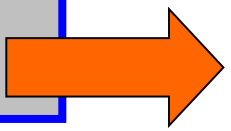
Implicit Objects

- `session`
  - The HttpSession object associated to the request
  - Same usage as in servlets
  - Created automatically
- `application`
  - The ServletContext object
  - Used to share variables across *all servlets* in the application
  - getAttribute and setAttribute methods
- `config`
  - The ServletConfig object
  - Same usage as in servlets
- `pageContext`
  - The PageContext object
  - Used for *sharing JavaBeans*

Request Parameters

- JSP provides access to the *implicit* object **request** that stores attributes related to the request for the JSP page as parameters, the request type and the incoming HTTP headers (cookies, referer, etc.).
- Example Request:
  - http://localhost/example.jsp?
    param1=hello&param2=world

```
<html>
<body>
<p><%= request.getParameter("param1") %></p>
<p><%= request.getParameter("param2") %></p>
</body>
</html>
```



```
<html>
  <body>
    <p>Hello</p>
    <p>World</p>
  </body>
</html>
```

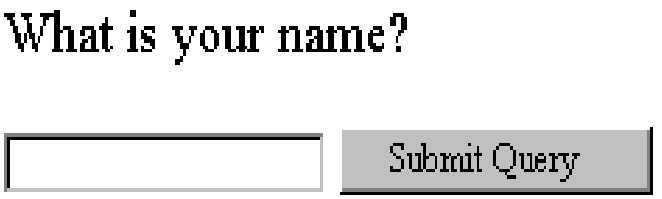
JSP Example: Hello World

- 1.

```
<html>
<head>
    <title>Hello World example</title>
</head>
<body>
Hello, World!
</body>
</html>
```

- 2.

```
<html>
<head>
    <title>Simple JSP Example</title>
</head>

<body>
<FORM METHOD="GET" ACTION="SimpleJSP.jsp">
<p> What is your name?</p>
<INPUT TYPE="TEXT" SIZE="20" NAME="name">
<INPUT TYPE="SUBMIT">
</FORM>

</body>
</html>
```



SimpleJSP.jsp

```
<html>
<head>
    <title>Simple JSP Example - version 1</title>
</head>

<body>
<P>
    <% String visitor = request.getParameter("name");
       if (visitor == null) visitor = "World";  %>
       Hello, <%=visitor%>!<BR>
</P>
</body>
</html>
```

Introduction to Java Server Pages

Part II

Advanced JSP tags and Java Beans

JSP Action elements

- Action elements are an important syntax element in JSP
- They are represented by tags (as is HTML)
- They assist JSP developers to develop in **tags** rather than **scriplet** programming
- Instead of <%, they just use the < character (like HTML)

```
<prefix:action_name>
    body
</prefix:action_name>
```

JSP Action elements

- JSP tags have a “start tag”, a “tag body” and an “end tag”
- The start and end tag have the same name enclosed in < and >
- The tag names have an embedded colon character “:” in them
  - the part before the colon (prefix) describes the type of the tag
  - the part after the “:” is the Action Name

```
<prefix:action_name>
    body
</prefix:action_name>
```

JSP Action elements

- Tags have associated **attributes** (like HTML e.g. <img src = "..")
- Full syntax of JSP Action Elements is:
  - `<prefix:action_name attr1 = "value" attr2 = "value2">`
    - `action_body`
  - `</prefix:action_name>`
- If the element doesn't have a body, can lose the end tag and use shorthand syntax of:
  - `<prefix:action_name attr1 = "value" attr2 = "value2" />`
- Example:
  - `<jsp:include page="scripts/login.jsp" />`

JSP Action Elements

- JSP Pre-defined tags
- Tag prefix: <jsp:...>
- Also called Standard Action Elements

- External tag library
  - JSTL
  - Custom tag library
- Tag prefix chosen by page developer

JSP Predefined Tags

- Also called JSP Standard Action Elements
  - <jsp:forward>
  - <jsp:include>
  - <jsp:param>
  - <jsp:plugin>
  - <jsp:useBean>
  - <jsp:getProperty>
  - <jsp:setProperty>
- See «JavaServer Pages™ Specification» for detailed attributes and values
  - http://jcp.org/aboutJava/communityprocess/final/jsr152/
  - http://java.sun.com/products/jsp/2.1/docs/jsp-2_1-pfd2/index.html

Standard JSP actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine.
- Available actions include:
  - **jsp:include** - Include a file at the time the page is requested.
  - **jsp:useBean** - Find or instantiate a JavaBean.
    - **jsp:setProperty** - Set the property of a JavaBean.
    - **jsp:getProperty** - Insert the property of a JavaBean into the output.
  - **jsp:forward** - Forward the requester to a new page.
  - **jsp:plugin** - Generate browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.

The jsp:forward Action

- This action lets you **forward the request** to another page.
- It has a single attribute, page, which should consist of a relative URL:
  - a static value
  - a string expression computed at request time.
- It *emulates* a new request from the browser

```
<jsp:forward page="/utils/errorReporter.jsp" />

<jsp:forward page="<%= someJavaExpression %>" />
```

Example

- Standard Action Example: <JSP: forward> tag
- **Stops** processing of one page and starts processing the page specified by the page attribute
- Example:

```
<html>
  <body>
    Error occurred…please wait<br/>
     <jsp:forward
  page=“errorpage.jsp"/>
  </body>
</html>
```

Forwarding with parameters

```
<jsp:forward page="urlSpec">
   <jsp:param name="param1Name"
    value="param1Value" />
   <jsp:param name="param2Name"
    value="param2Value" />
   ...
</jsp:forward>
```

The jsp:include Action

- Unlike the include directive, which inserts the file at the time the JSP page is translated into a servlet, this action inserts the file **at the time the page is requested:**
  - Small penalty in efficiency
  - The included page cannot contain JSP code (only HTML)
  - Gains significantly in flexibility.

The <jsp:include> Action

- Standard Action Example: <jsp:include> tag
- Example:

```
<html>
   <body>
     Going to include hello.jsp...<br/>
     <jsp:include page="hello.jsp"/>
   </body>
</html>
```

- Executes the included JSP page and adds its output into the page

Include vs. Include

- What's the difference from using the 'include' directive?
  - `<%@ include file = 'hello.jsp' %>`
- The include directive includes the contents of another file **at compilation time**.
  - Good for including common static code e.g. header file, footer file.
  - Good on performance: included only once.
- But, what if including dynamic common code (e.g. a navigation bar where links are read from the DB?).
  - Need to re-run the file each time a request is made - use jsp:include
  - `jsp:include` incorporates the output of the included JSP file **at run time**

jsp:param with jsp:include

- Can be used to pass parameters when using <jsp:include> or <jsp:forward>
- Example

```
<jsp:include page="login.jsp">
   <jsp:param name="user" value="smith" />
</jsp:include>
```

  - Executes a login page
  - jsp:param passes in username to the login page

Java Beans

- Java Beans are reusable components. They are used to separate Business logic from the Presentation logic.
- Internally, a bean is just an instance of a class.
- JSP's provide three basic tags for working with Beans:
  - <jsp:useBean >
  - <jsp:setProperty>
  - <jsp:getProperty>

The BEAN structure

- The Java BEAN is not much different from a Java program.
- The main differences are the signature methods being used in a bean.
- For passing parameters to a bean, there has to be a corresponding get/set method for every parameter.
- The class should be serializable (able to persistently save and restore its state)
- It should have a no-argument constructor

The jsp:useBean Action

- This action lets you load in a JavaBean to be used in the JSP page.
- This is a a very useful capability because it lets you exploit the reusability of Java classes without sacrificing the convenience that JSP adds over servlets alone.
- The simplest syntax for specifying that a bean should be used is:

```
<jsp:useBean id="name"
class="package.class" />
```

Java Beans

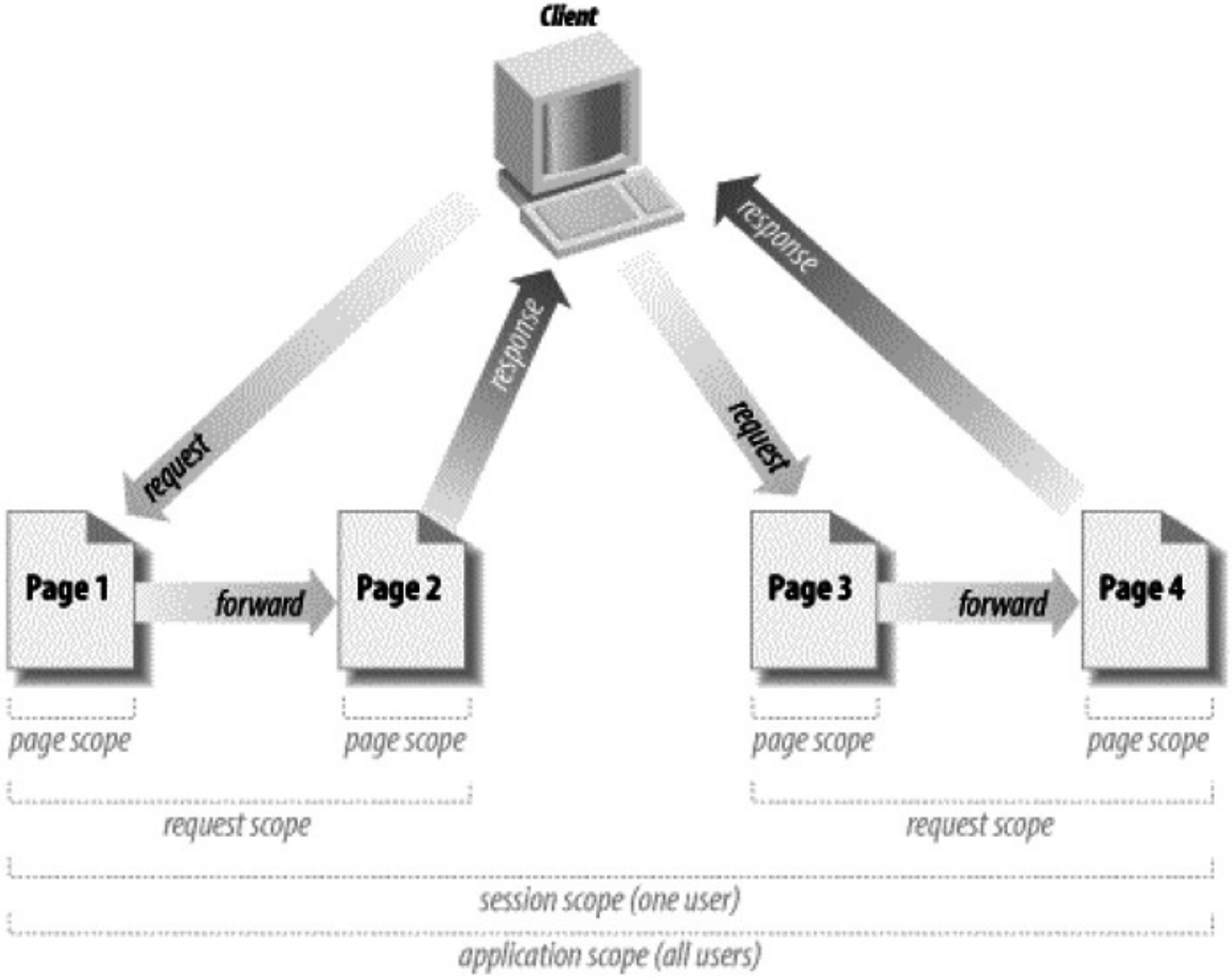
- To use a bean in a JSP page, three attributes must be supplied
  - an **id**, which provides a local name for the bean
    - Creates a “variable” used to access the bean
  - the bean's **class name**, which is used to instantiate the bean if it does not exit
    - Suggestion: always use **packages** to help Tomcat find the class!
  - a scope, which specifies the lifetime of the bean.

```
<jsp:useBean id="bean name"
             class="bean class"
scope = "page | request | session |
             application" />
```

Bean Scopes

- There are four scopes available: page, request, session, and application.
  - A **page**-scoped bean is available only within the JSP page and is destroyed when the page has finished generating its output for the request. By default all beans have page scope
  - A **request**-scoped bean is destroyed when the response is sent.
  - A **session**-scoped bean is destroyed when the session is destroyed.
  - An **application**-scoped bean is destroyed when the web application is destroyed.

Bean Scopes


Client
request
response
request
response
Page 1
forward
Page 2
Page 3
forward
Page 4
page scope
page scope
page scope
page scope
request scope
request scope
session scope (one user)
application scope (all users)

jsp:setProperty / jsp:getProperty

- You use jsp:setProperty to give values to properties of beans that have been referenced earlier
  - By default the values in jsp:setProperty is taken from a parameter in the request with the same value.
- You use jsp:getProperty to retrieve the value of a bean property, convert it to a string, and to insert it into the output.

You must use a <jsp:useBean> tag to declare the Bean before you can use <jsp:setProperty>

```
<jsp:useBean id="itemBean" ... /> ...
<ul>
<li>Number of items:
  <jsp:getProperty name="itemBean"
  property="numItems" /></li>
<li>Cost of each:
  <jsp:getProperty name="itemBean"
  property="unitCost" /></li>
</ul>
```

jsp:setProperty

- `<jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />`
  - Sets the property of the given bean to the specified value
  - beanName must be the same name used in the id of jsp:useBean

- `<jsp:setProperty name="beanName" property="propertyName" value="<%= expr %>" />`
  - Uses a run-time expression to set a property value

jsp:setProperty

- `<jsp:setProperty name="beanName" property="propertyName" param="parameterName" />`
  - Sets the property to the value of a Request parameter (HTML form)
  - If the parameter is not present, or if it is empty, no action is taken

- `<jsp:setProperty name="beanName" property="propertyName" />`
  - Sets the property from a parameter name with the *same name* of the property name

jsp:setProperty

- `<jsp:setProperty name="beanName" property="*" />`
  - Automatically tries to set all (not-empty) Request parameters

jsp:getProperty

- `<jsp:getProperty name="beanName" property="propertyName" />`
  - Gets the property from the given bean
  - beanName must be the same name used in the id of jsp:useBean
  - The value will be converted to a String and inserted in the HTML page

SimpleJSP.jsp - the Bean edition

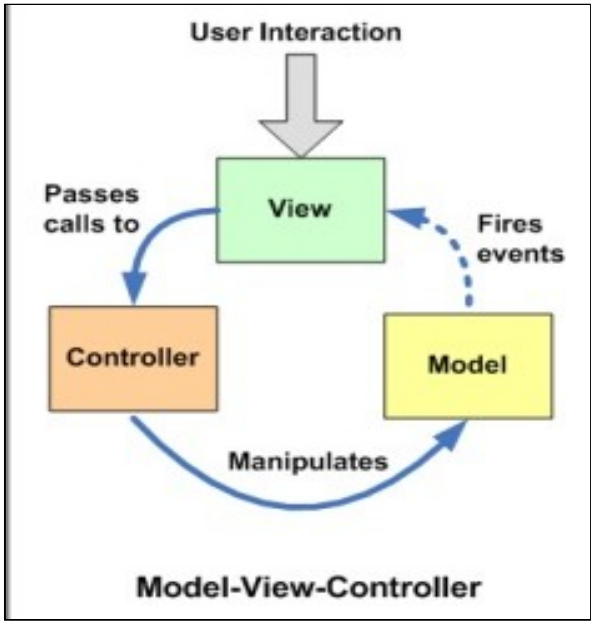
```
package examples.HelloBean;

public class HelloBean implements java.io.Serializable
{
    String name;

    public HelloBean ()
    {
        this.name = "World";
    }

    public String getName ()
    {
        return name;
    }

    public void setName (String name)
    {
        this.name = name;
    }

}
```

SimpleJSP.jsp - the Bean edition

```
<html>
<head>
    <title>Simple JSP Example - version 2</title>
</head>

<body>
<jsp:useBean id="hello" class="examples.HelloBean"/>
<jsp:setProperty name="hello" property="name" param="name"/>
<P>
Hello, <jsp:getProperty name="hello" propety="name"/>!<BR>
</P>
</body>
</html>
```

MVC design pattern

- A web application:
  - Collects data and action requests from users…
  - …elaborates/stores them…
  - …visualize the results

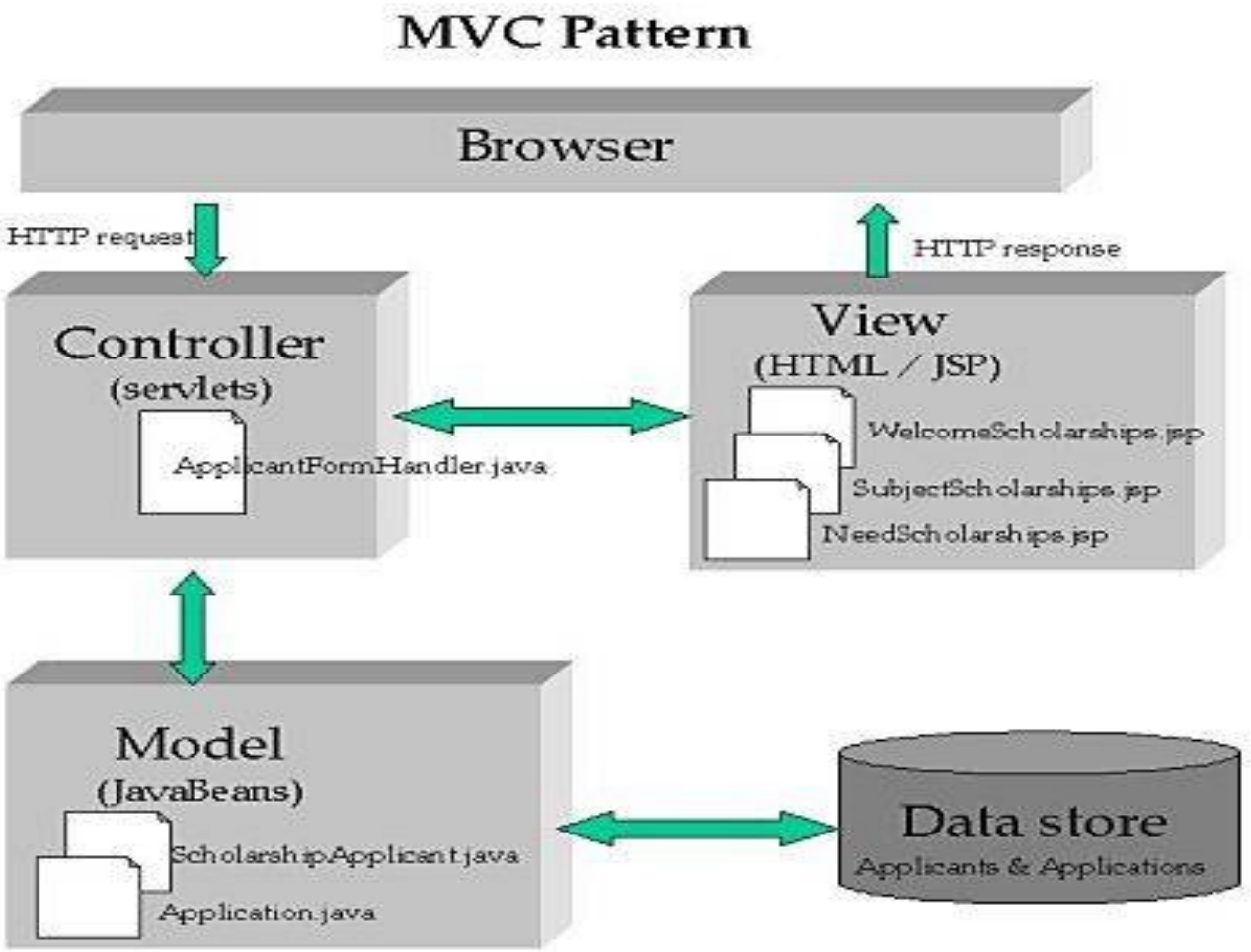


- MVC – Model View Controller paradigm
- The **model** represents the current state of the applications (with respect to a finite state machine)
- The **view** corresponds to a presentation of the state
- The **controller** verifies collected data and updates the model

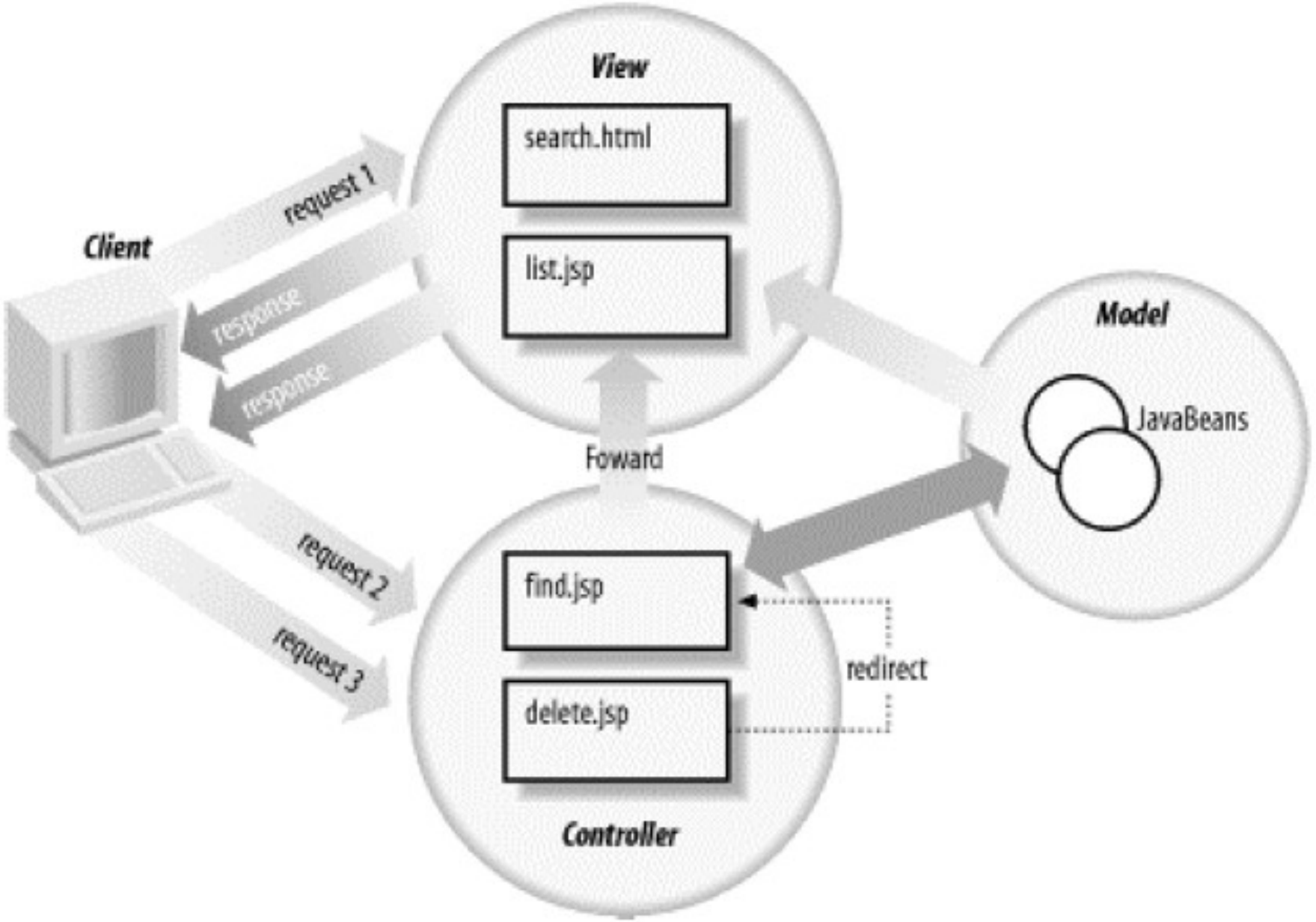
MVC

- Applications that present lots of data to the user, often wish to separate data (**Model**) and user interface (**View**) concerns
- Changing the user interface do not impact the data handling, and that the data can be reorganized without changing the user interface.
- The MVC design pattern solves this problem by decoupling data access and business logic from data presentation and user interaction.

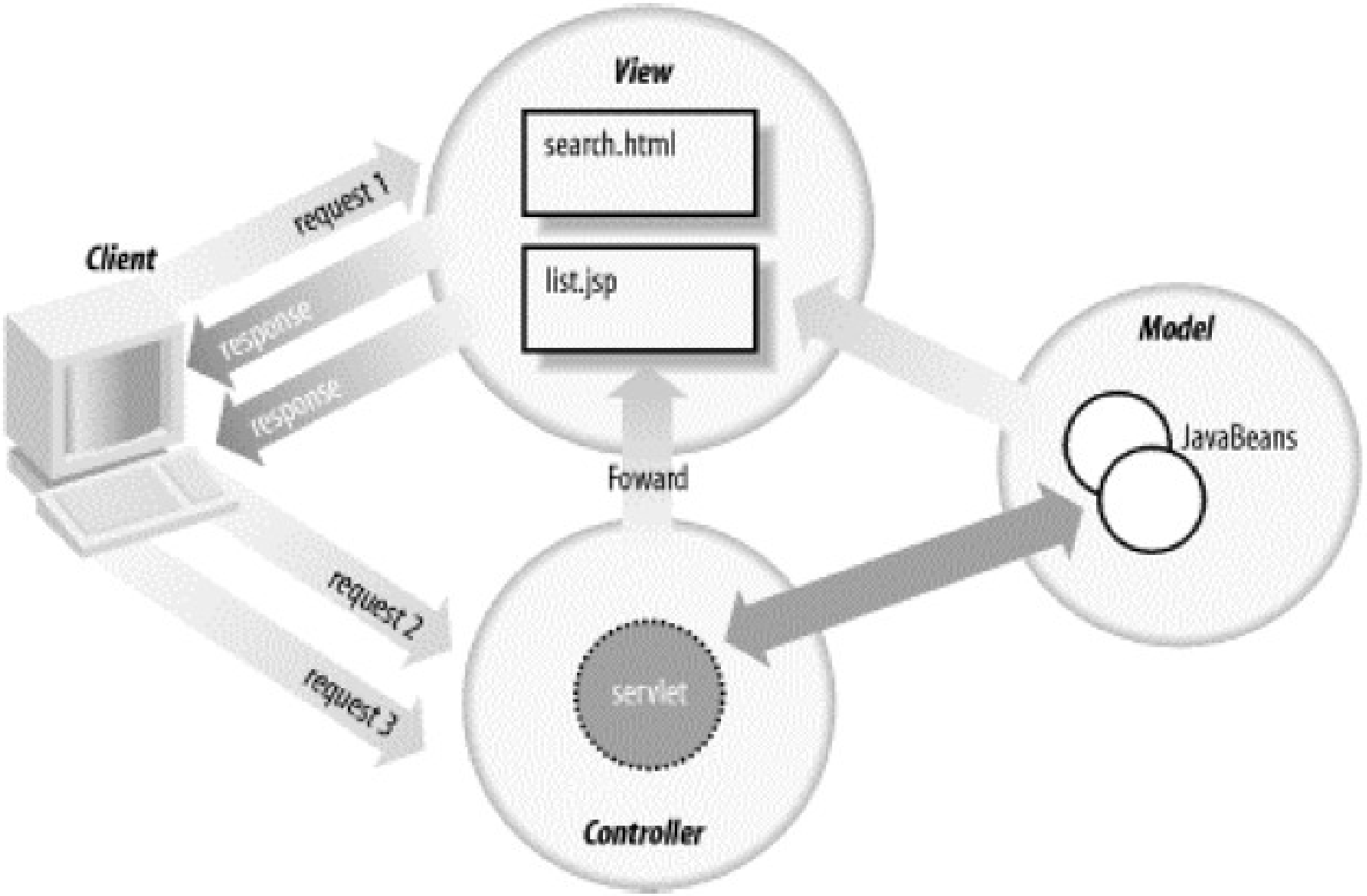
MVC in the Java Server architecture



MVC with JSP only



MVC with JSP and servlets



MVC in J2EE: JSP, Servlet, EJB