Fpt University

Java Server Pages

JSP Syntax

Techniques:

MVC Design Pattern

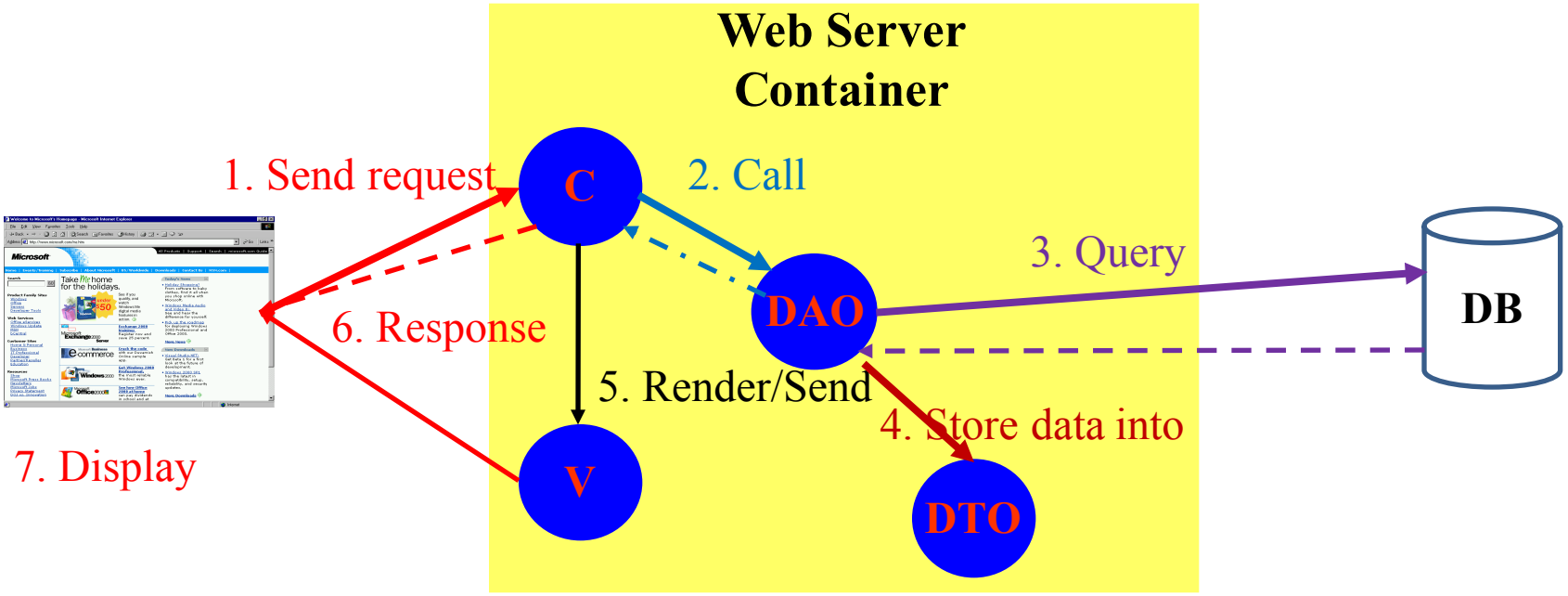
Dynamic DB Connection

#JSP #MVC #JavaEE

Review

- **How to deploy the Web Application to Web Server?**
  - File and Directory Structures (WEB-INF and others → **war** file)
  - The Servlet Container, The Servlet Context, The Servlet Config
    - Parameters
- **How to transfer from resources to others with/without data/objects?**
  - **Attributes**
    - Vs. Parameters, Vs. Variables
    - Request, Session, and Context Scope (Memory Segment)
  - **Request Dispatcher**
    - forward, include
    - Vs. response.sendRedirect
  - Break down structure component in building web application
- **Some concepts**
  - **Filter**
    - Filter, Filter Chain, Filter with Wrapper class
    - Vs. Request Dispatcher

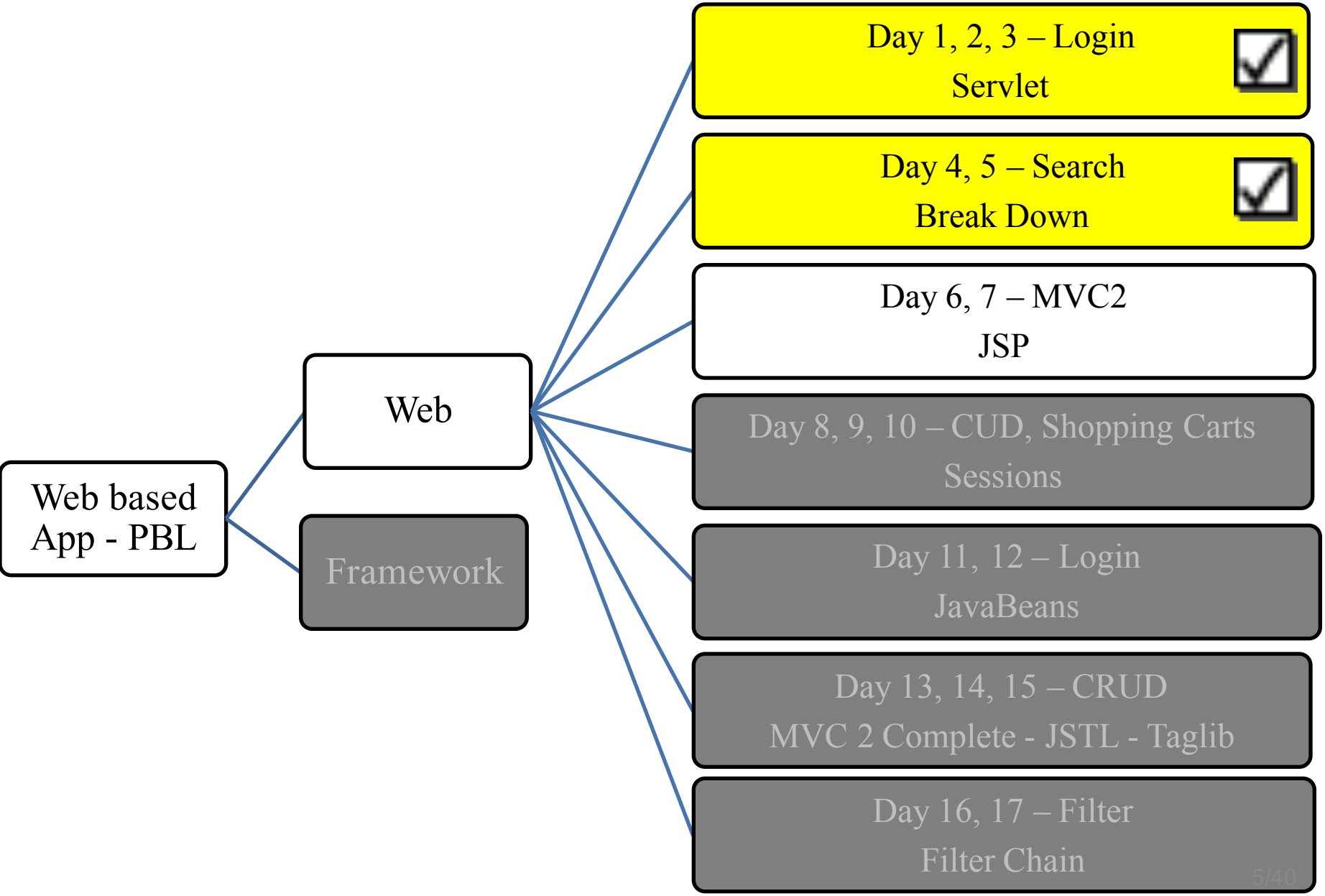
Review


Web Server
Container
1. Send request
C
2. Call
3. Query
DB
DAO
6. Response
5. Render/Send
4. Store data into
7. Display
V
DTO

Objectives

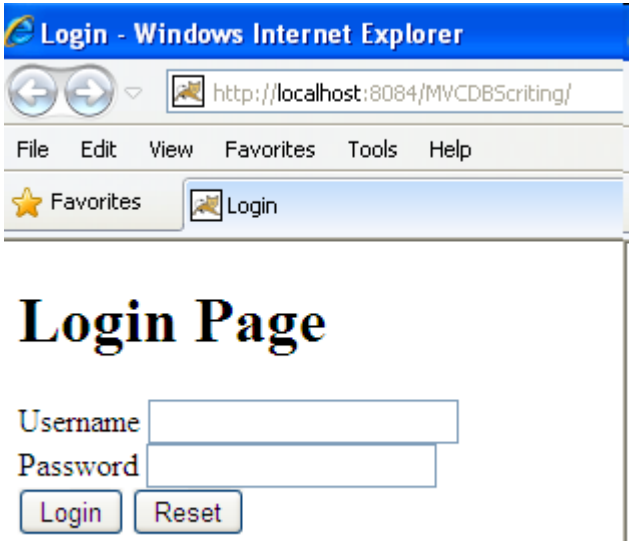
- **How to build web application applying MVC model using Servlet, JSP + Scripting Element**
  - MVC Model
  - JSP vs. Servlet
  - JSP mechanism, syntax
  - How to use JSP combining the Servlets and Java objects
  - How to connect DB using Dynamic connection or DataSource

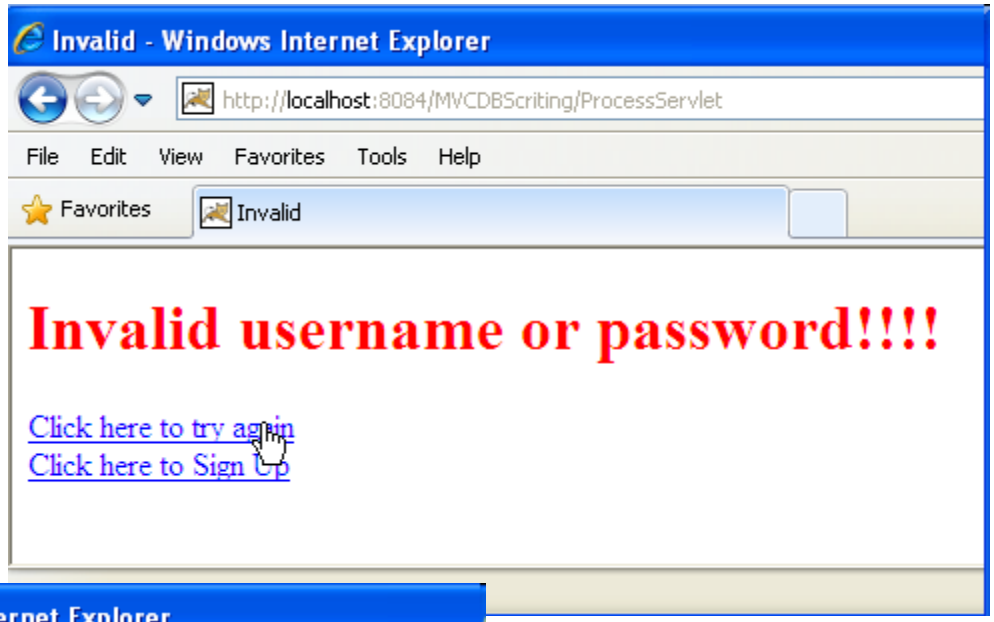
Objectives

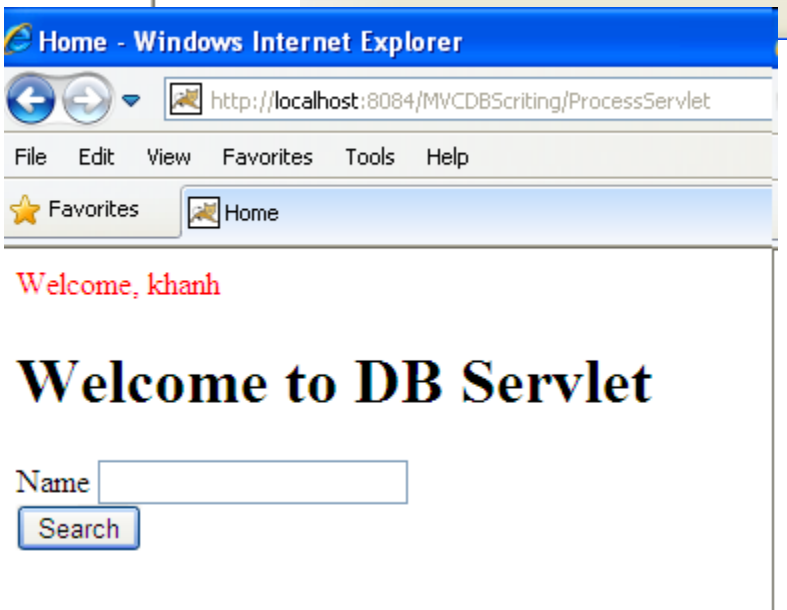

Web based App - PBL
Web
Framework
Day 1, 2, 3 – Login Servlet
Day 4, 5 – Search Break Down
Day 6, 7 – MVC2 JSP
Day 8, 9, 10 – CUD, Shopping Carts Sessions
Day 11, 12 – Login JavaBeans
Day 13, 14, 15 – CRUD MVC 2 Complete - JSTL - Taglib
Day 16, 17 – Filter Filter Chain

MVC Design Pattern

MVC Model 2







MVC Design Pattern

MVC Model 2

Welcome, khanh

Search Page

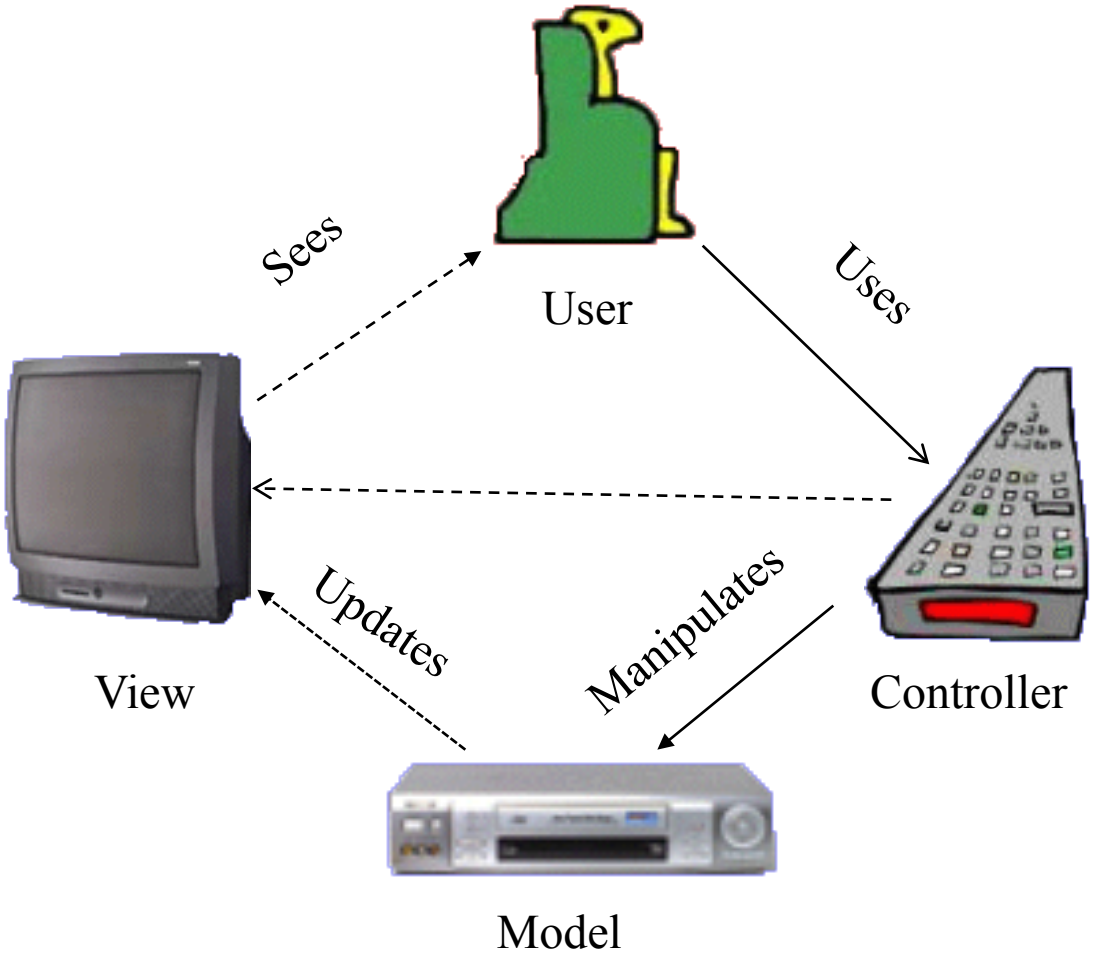
Search Value a

Search

No.	Username	Password	Last name	Role
1	IA1161	123456	Class IA1161	☐
2	khanh	kieu123	Khanh Kieu	☑
3	SE1161	123456	Class SE1161	☐
4	SE1162	123456	Class Se1162	☐
5	SE1163	123456	Class SE1163	☐

MVC Design Pattern

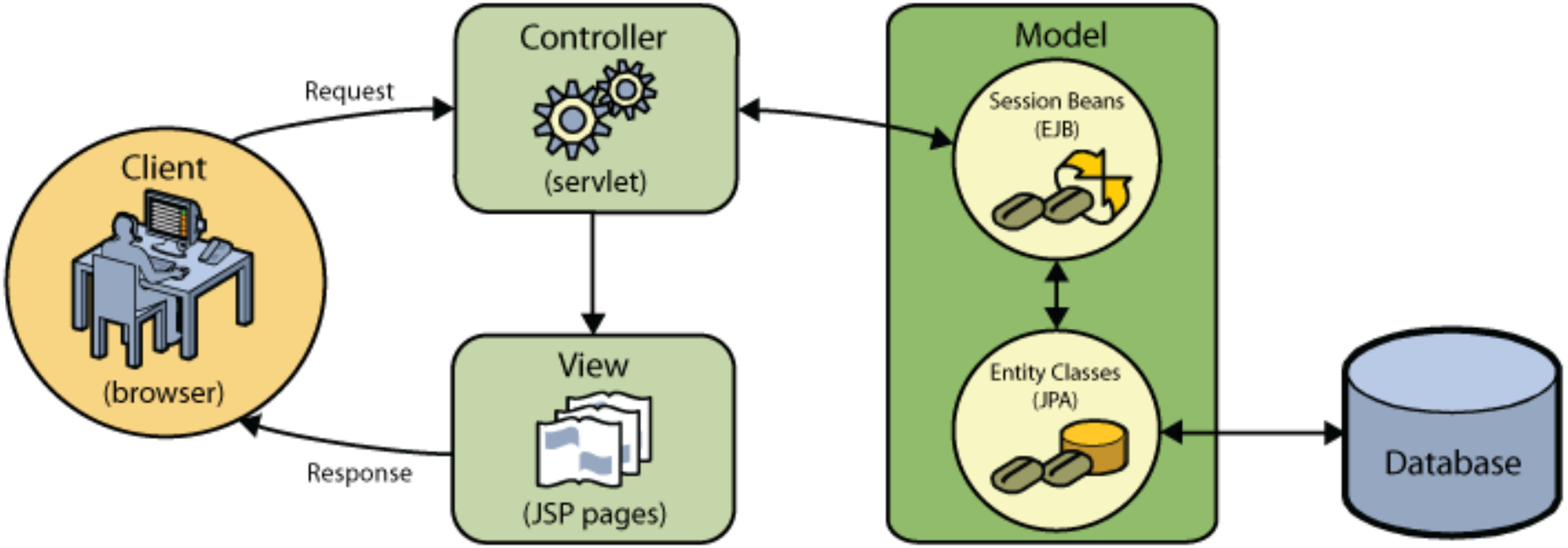
Model – View – Controller



This is a MVC Model

MVC Design Pattern

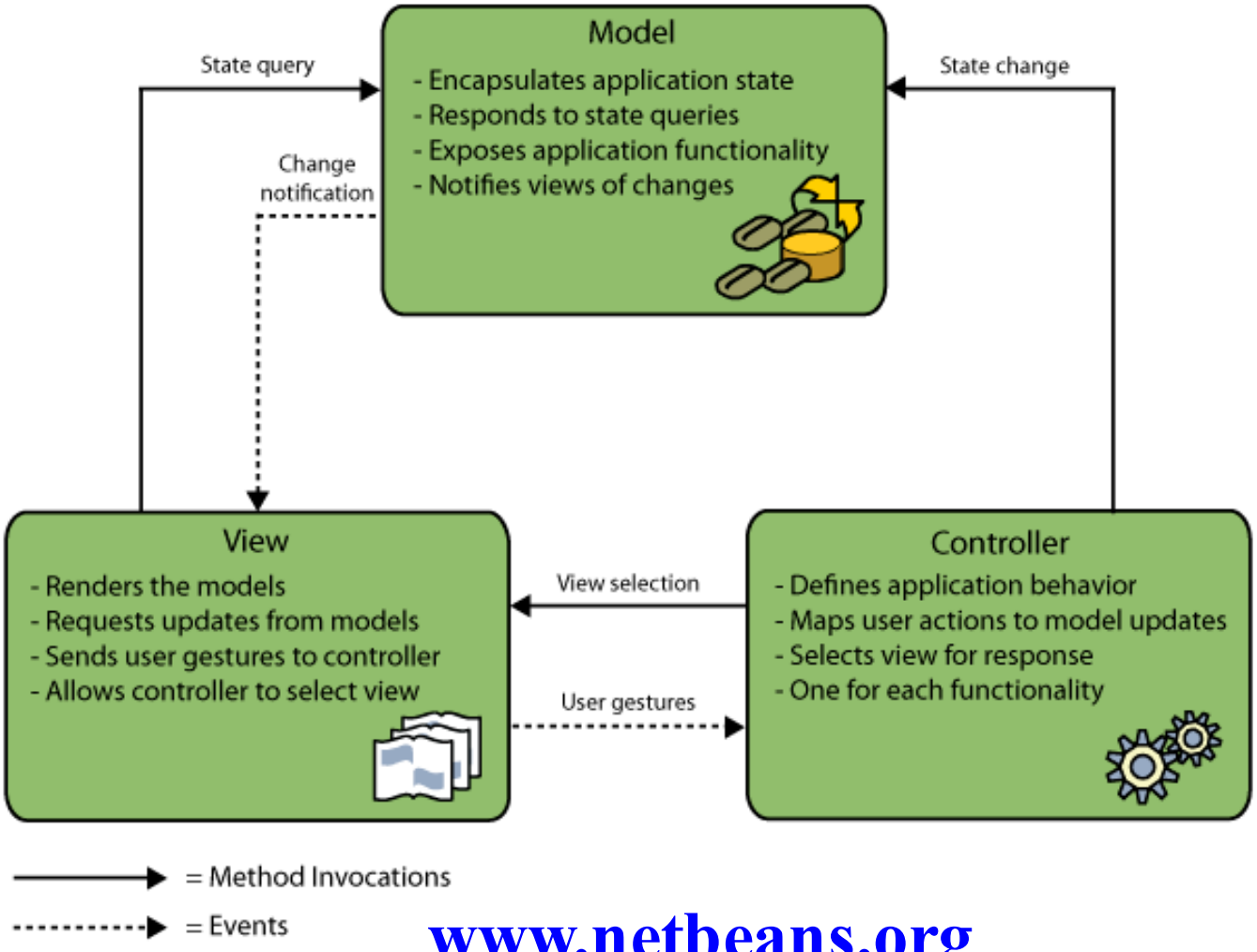
Model – View – Controller



www.netbeans.org

MVC Design Pattern

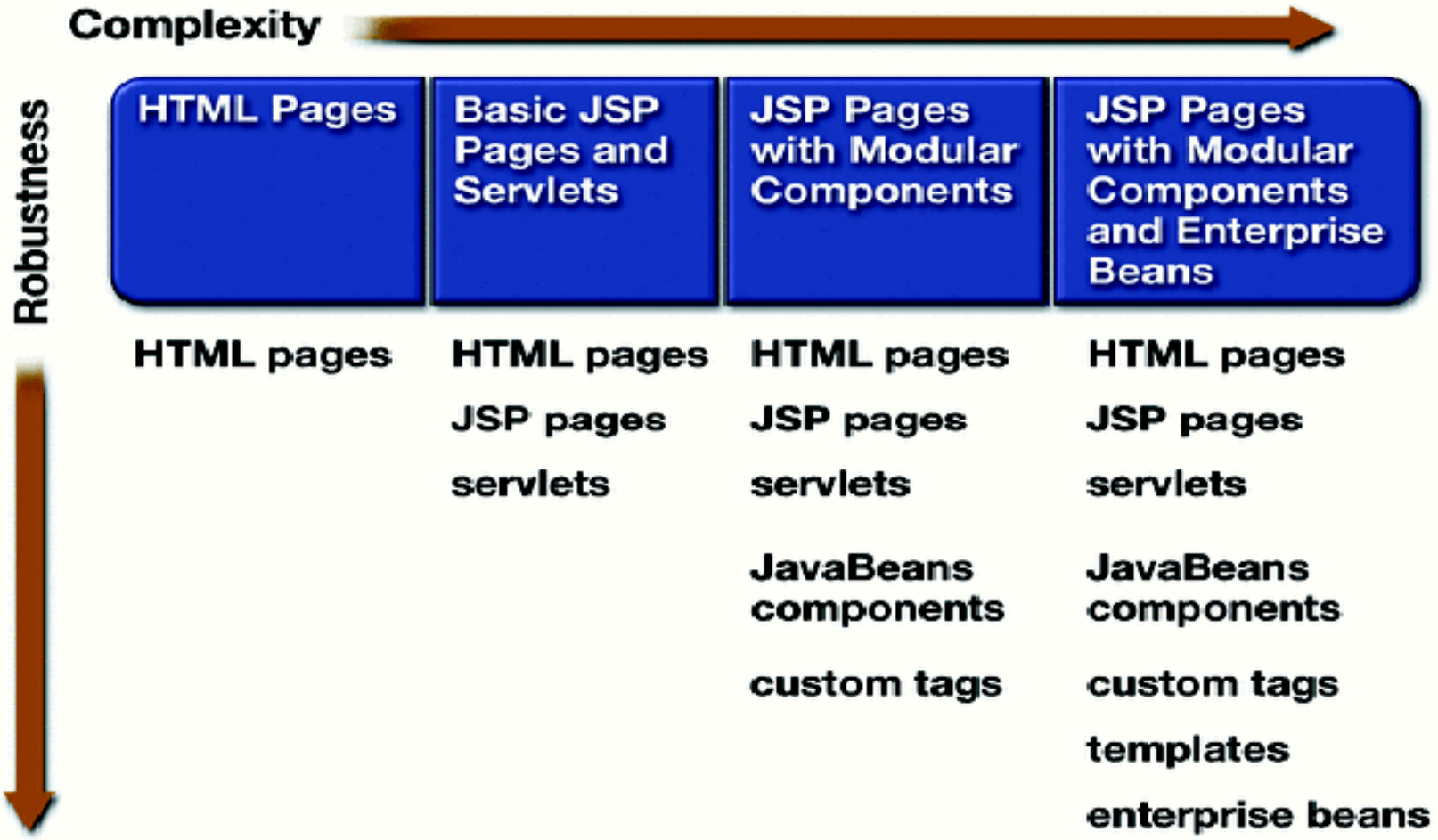
Model – View – Controller



www.netbeans.org

MVC Design Pattern

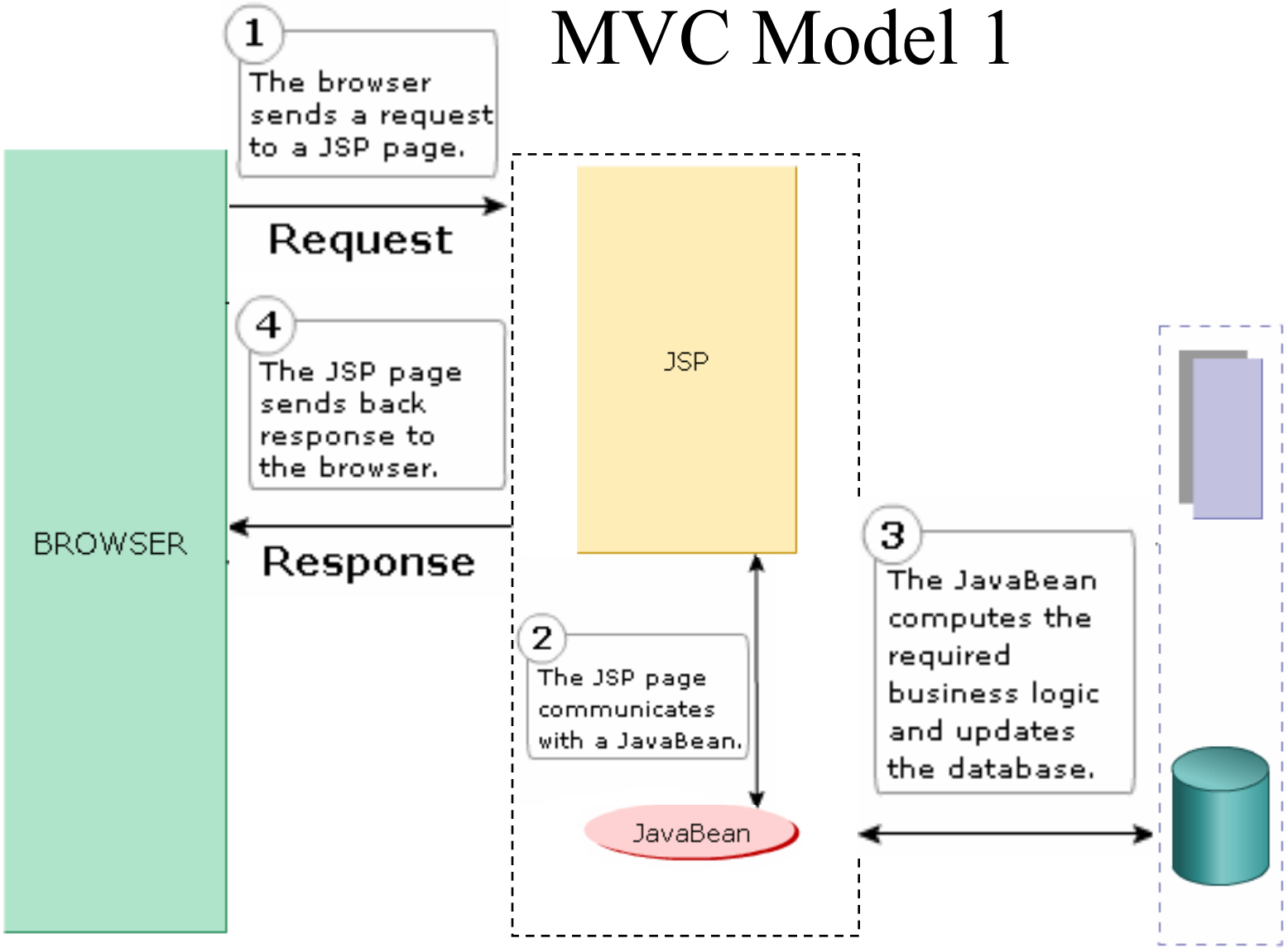
No MVC



NO MVC	Model 1 Architecture

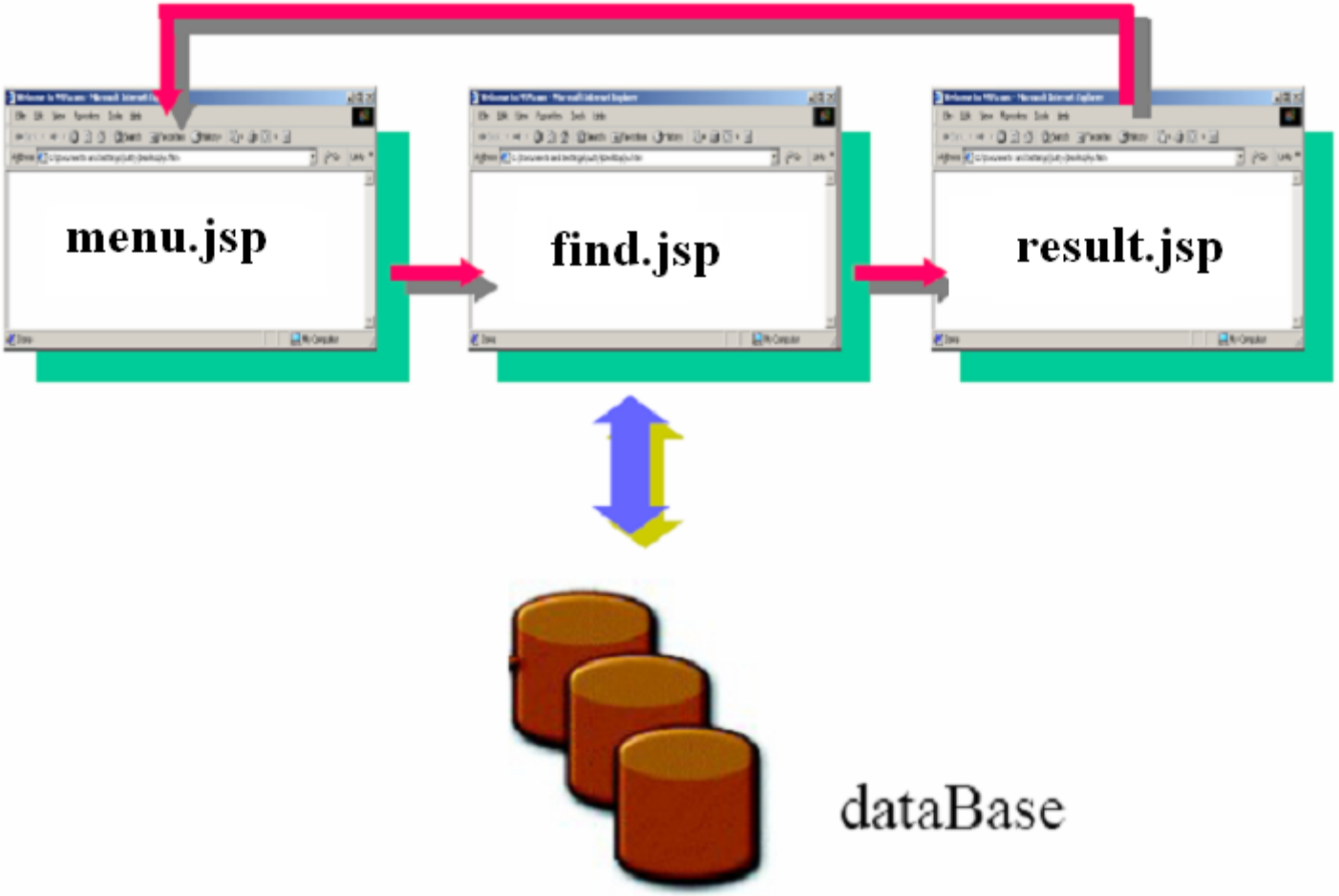
MVC Design Pattern

MVC Model 1



MVC Design Pattern

MVC Model 1



Page-centric application

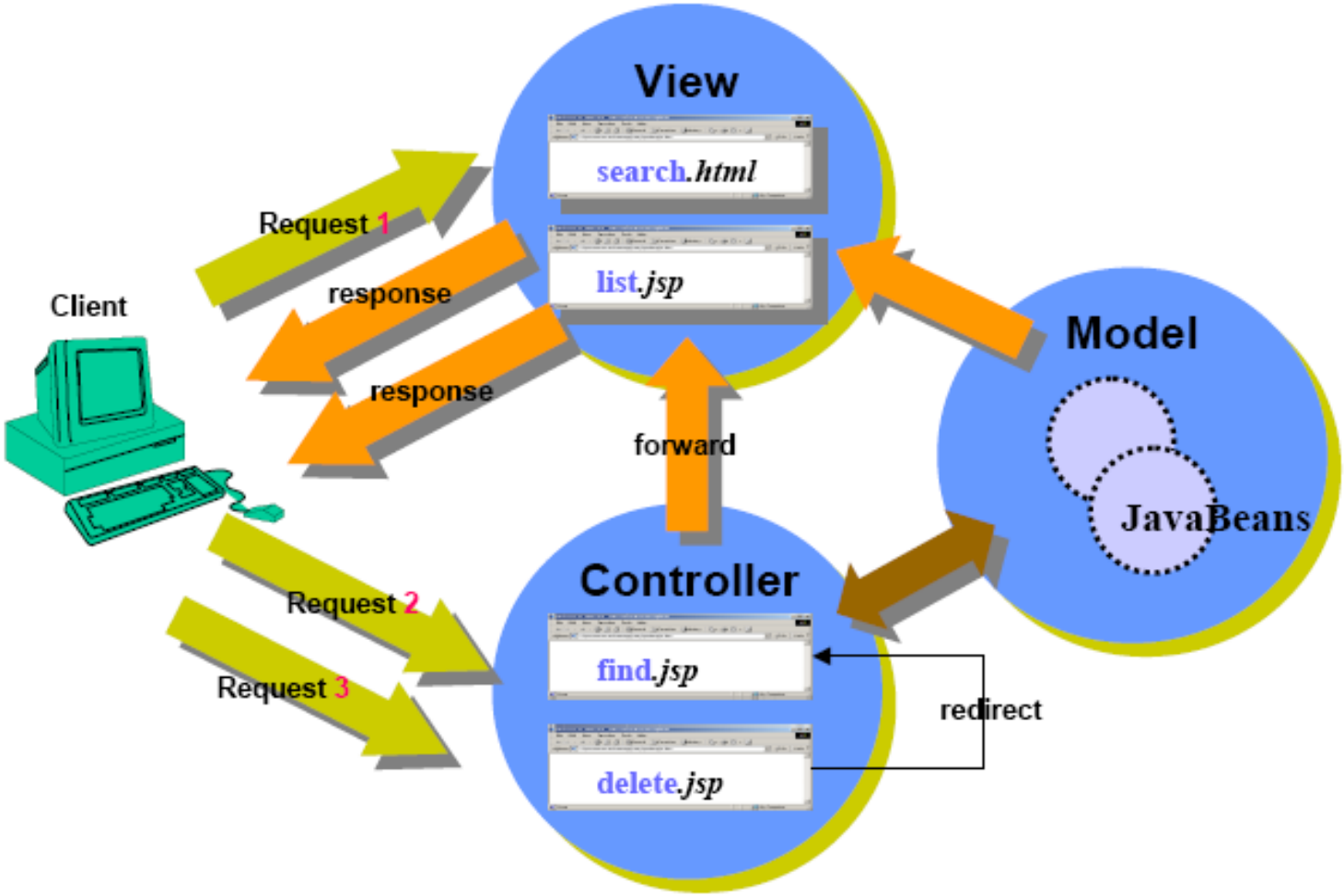
MVC Design Pattern

MVC Model 1 – Example

Example will be shown in other topic

MVC Design Pattern

MVC Model 1 – Generalization



Page-centric Scenario

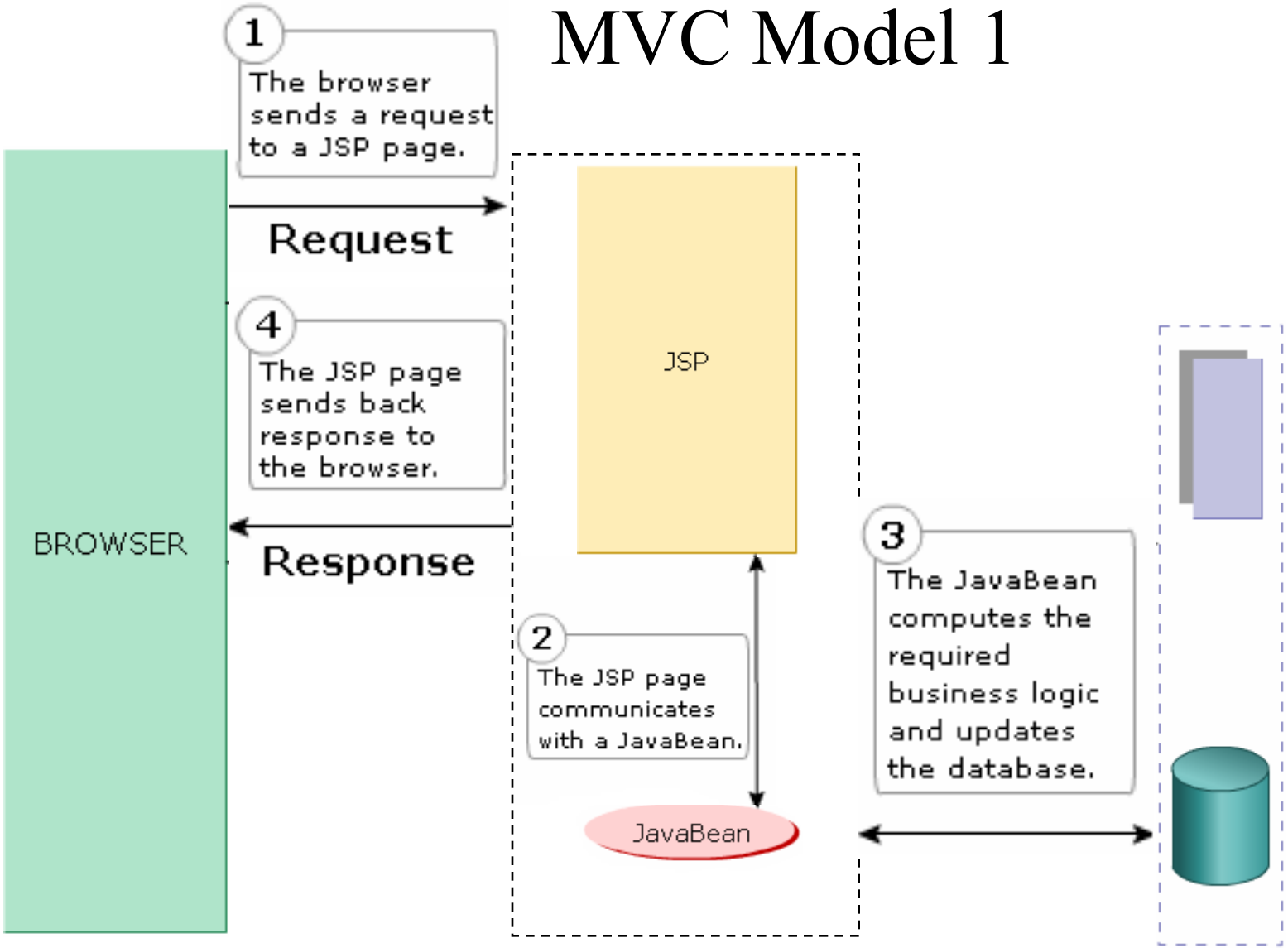
MVC Design Pattern

MVC Model 1 – Generalization

- **Purpose**
  - Separate "business logic" from "presentation logic"
  - Need for **centralized security control** or logging, changes little over time.
  - Apply to applications that have very simple page flow.
- **Advantages**
  - Lightweight design – for **small, static application**
  - Separation of presentation from content
- **Limitations**
  - **Navigation Problem** – Changing name of JSP file must change in many location
  - **Difficult** to **maintain** an application – large java code being embedded in JSP page
  - **Inflexible**
  - **Performance** is **not high**
  - **Not scale** up over a period time
  - **Not Suitable for large and complex application**

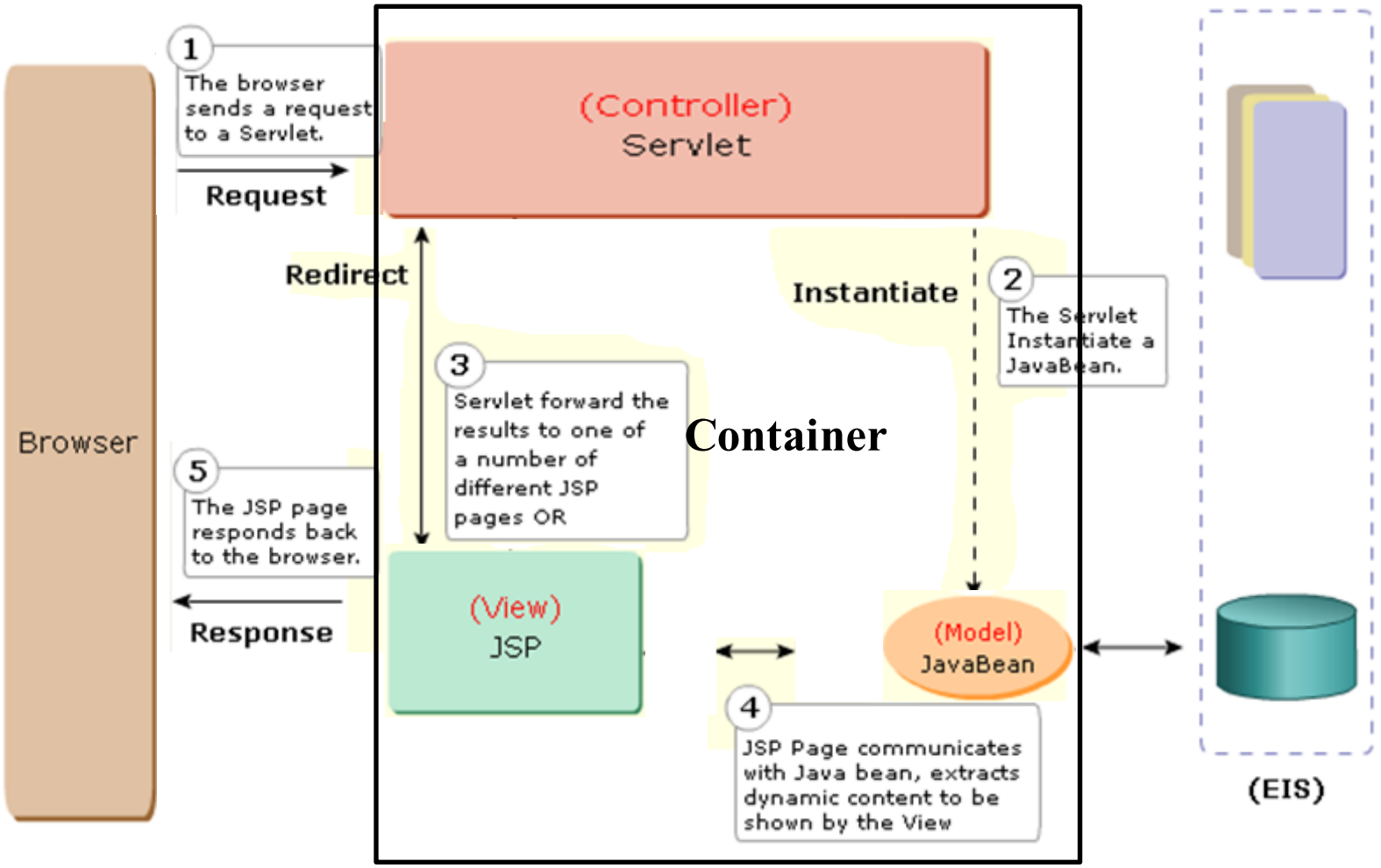
MVC Design Pattern

MVC Model 1



MVC Design Pattern

MVC Model 2



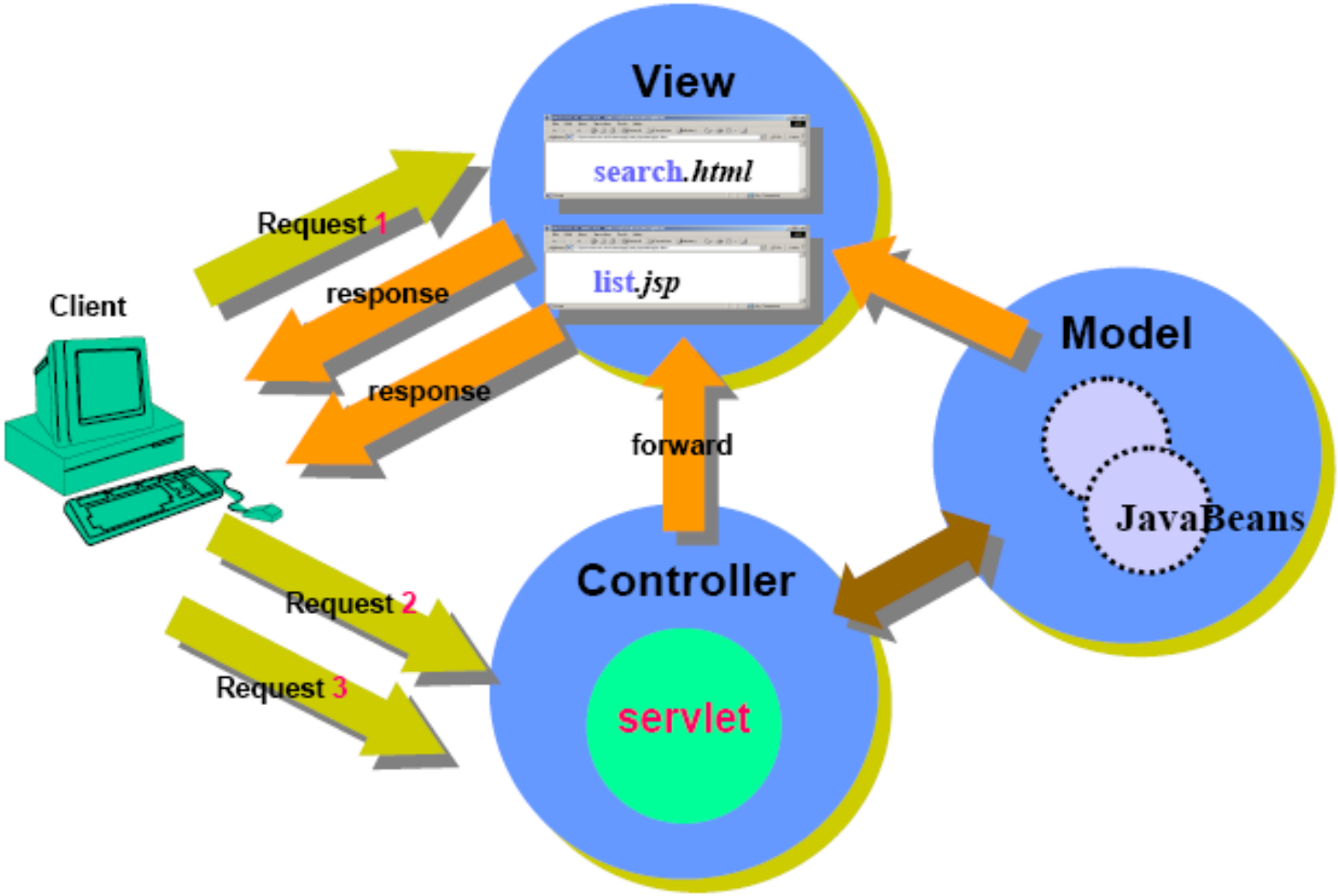
MVC Design Pattern

MVC Model 2

- **Separates** the "**Business Logic**" from the "**Presentation Logic**" and has an additional component – a Controller
- **Use Servlet** and **JSP** together
- **JSP pages**
  - Are used only for **presentation**
  - **Retrieve** the **objects** created by the Servlet
  - **Extract dynamic content** for insertion within a template for display
- **Servlet**
  - **Handles** initial **request**, partially **process** the **data**, **set up beans**, **select** suitable business **logic** to handle request , then **forward** the **results** to one of a number of **different JSP pages**
  - **Serves** as a **gatekeeper** provides **common services**, such as authentication authorization, login, error handling, and etc
  - Servlet **serves** as a **central controller** that act as a state machine or an event **dispatcher** to decide upon the appropriate logic to handle the request
  - Act as a **Controller** that controls the way Model and View layer interacts

MVC Design Pattern

MVC Model 2 – Generalization



Servlet-centric Scenario

MVC Design Pattern

MVC Model 2 – Generalization

- **Purpose**
  - **Separation** of presentation logic and business logic
  - A **Model** represents information specific to a particular domain on which the application operates
  - A **View** is typically a user interface element. A single Model may be presented as multiple views
  - A **controller** component responds to events and processes request and may invoked changes on the Model
- **Advantages**
  - **Easier** to build, maintain and extend
  - Provide **single point** of control (Servlet) for security & logging
  - **Encapsulate** incoming data into a form usable by the backend
  - Can **reusable code**
- **Limitations**
  - Increase Design Complexity

MVC Design Pattern

MVC Model 1 & Model 2 Comparison

Criteria	Model 1	Model 2
JSP page responsibility	Processing requests and sending back replies to clients	Are used for the presentation layer
Servlets responsibility	N/A	Are used for processing task
Type of Web application	Simple	Complex
Nature of Developer's Task	Quick prototyping	Developing an application that can be modified and maintained
Who is doing the work?	View and Controller is developed by the same team	View and Controller is developed by the different teams

Java Server Pages
Need for JSP

- Servlet
  - Is a java class that **must be compiled** to **deploy** on server
  - Low level HTML format that does **not focus** on the **presentation logic (very complex)** of the web application
  - Is **not flexible** in modify with editor program

→ **Need** the **replaced** thing that is easily focus on the **presentation logic** and **approaches** the **non-experience** presentation developer

Java Server Pages

JSP

- Java Server Page (JSP) is a **server side script language** running web (application) server (Tomcat, Sun, JBoss …)
- Saved with **.jsp extension**
- A simple, yet powerful Java technology for **creating and maintaining dynamic-content** webs pages (embedded)
- JSP page **are converted** by the web container **into a Servlet instance**
- It focus on the **presentation logic** of the web application
- JSP page **contains HTML tags**
- JSP page contains tags (standard & custom), which are used to generate dynamic content and invoke the operations on Javabeans components and processing requests.
- **A combination of HTML, XML, Servlet** (extends from Servlet), **and Java Code** to create dynamic Web content.

Java Server Pages

JSP

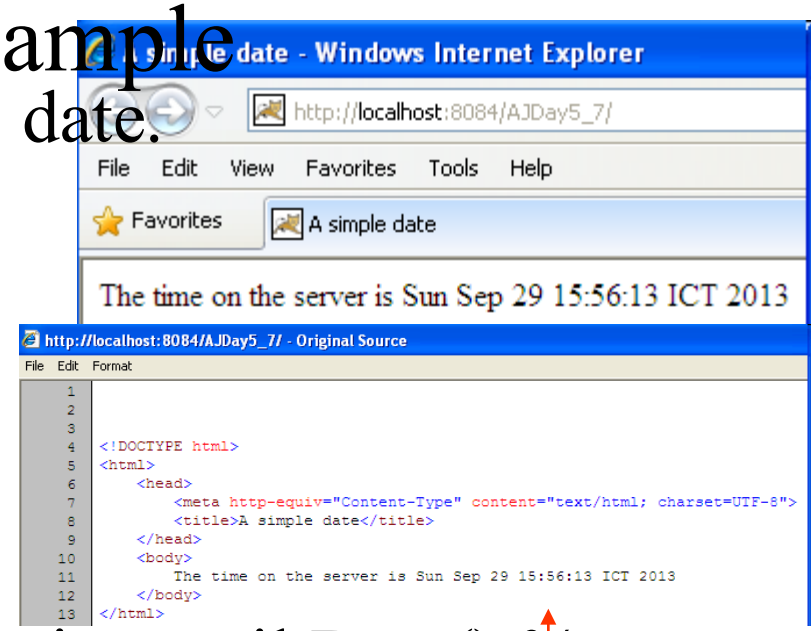
- Benefits
  - Segregation of the work profiles of a Web designer and a Web developer (**separating presentation logic and content/business/processing logic**)
  - Emphasizing Reusable Components (**JavaBeans)**
  - Simplified Page Development (**easy to use JSP through tag, flexibility, scalability**)
  - Access & instantiate JavaBeans component (**support tag element with get/set functions**)
  - High secure
- **Choosing Servlet or JSP**
  - **Servlet** are well suited for **handling binary data dynamically**
    - Ex: for uploading files or for creating dynamic images, since they **need not contain** any **display logic**
  - **JSP** is **easy** to **create** the **presentation logic with dynamic generated data** combine template and **do not compile before executing or running time**

Java Server Pages

JSP – Example

- **Simple JSP page** displays current date.

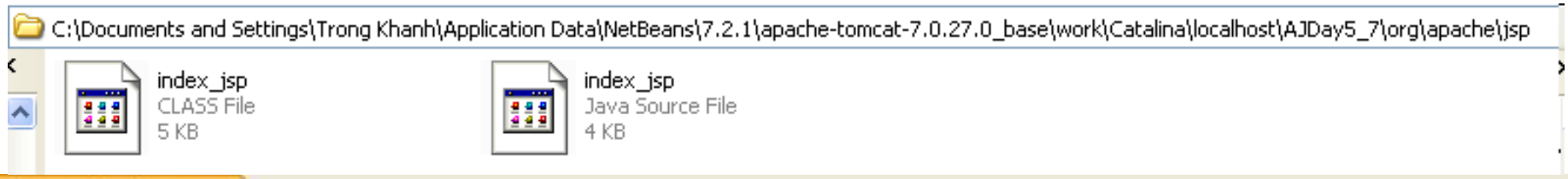
```
<html>
  <head>
    <title>A simple date</title>
  </head>
  <body>
    The time on the server is <%= new java.util.Date() %>
  </body>
</html>
```



- Server processes JSP components converting static data on **HTML** which can be displayed by Web browser
- To testing JSP page, the JSP page should be copied to the ROOT of web server – Tomcat

Java Server Pages

JSP – In Nature



index_jsp.java

Source | History

```
62          out.write("\n");
63          out.write("\n");
64          out.write("\n");
65          out.write("<!DOCTYPE html>\n");
66          out.write("<html>\n");
67          out.write("    <head>\n");
68          out.write("        <meta http-equiv=\"Content-Type\" content=\"text/
69          out.write("        <title>A simple date</title>\n");
70          out.write("    </head>\n");
71          out.write("    <body>\n");
72          out.write("        The time on the server is: ");
73          out.print( new java.util.Date());
74          out.write("\n");
75          out.write("    </body>\n");
76          out.write("</html>\n");
77          out.write("\n");
```

Applying the compile function on file to check JSP file

```
public void
_jspService(HttpServletRe
quest request,
HttpServletResponse
response)
      throws
java.io.IOException,
ServletException {
```

Java Server Pages

JSP – In Nature

- When the JSP page is requested to server, the JSP page is converted to java file as **filename_jsp.java** (**filename.jsp**)
- The filename_jsp.java file is **complied** if it is correct syntax
- **Ex**:
  - Omit the ")" of Date function in the simpleDate.jsp
  - Correct above mistake, run the file, then checking the result at
    - **C:\Documents and Settings\LoggedUser\Application Data\NetBeans\7.4\apache-tomcat-7.0.41.0_base\work\Catalina\localhost**
    - **C:\Users\LoggedUser\AppData\Roaming\NetBeans\7.4\apache-tomcat-7.0.41.0_base\work\Catalina\localhost**

```
Z:\LapTrinh\Servlet\AJ\AJDay5_7\build\generated\src\org\apache\jsp\index_jsp.java:55: error: ')' expected
      out.print( new java.util.Date( );
                                     ^
1 error
1 warning
Z:\LapTrinh\Servlet\AJ\AJDay5_7\nbproject\build-impl.xml:949: The following error occurred while executing this line:
Z:\LapTrinh\Servlet\AJ\AJDay5_7\nbproject\build-impl.xml:941: The following error occurred while executing this line:
Z:\LapTrinh\Servlet\AJ\AJDay5_7\nbproject\build-impl.xml:321: Compile failed; see the compiler error output for details.
BUILD FAILED (total time: 2 seconds)
```

Java Server Pages

JSP

index.jsp

Source | History

```
4       Author      : Trong Khanh
5   --%>
6
7   <%@page contentType="text/html" pageEncoding="UTF-8"%>
8   <!DOCTYPE html>
9   <html>
10      <head>
11          <meta http-equiv="Content-Type" content="text
12          <title>A simple date</title>
13      </head>
14      <body>
15          The time on the server is <%= new Date()%>
16      </body>
17  </html>
```

Apache Tomcat/7.0.34 - Error report - Windows Internet Explorer

http://localhost:8084/AJDay5_7/

File Edit View Favorites Tools Help

Favorites | Apache Tomcat/7.0.34 - Error report

HTTP Status 500 - Unable to compile class for JSP

type Exception report

message Unable to compile class for JSP:

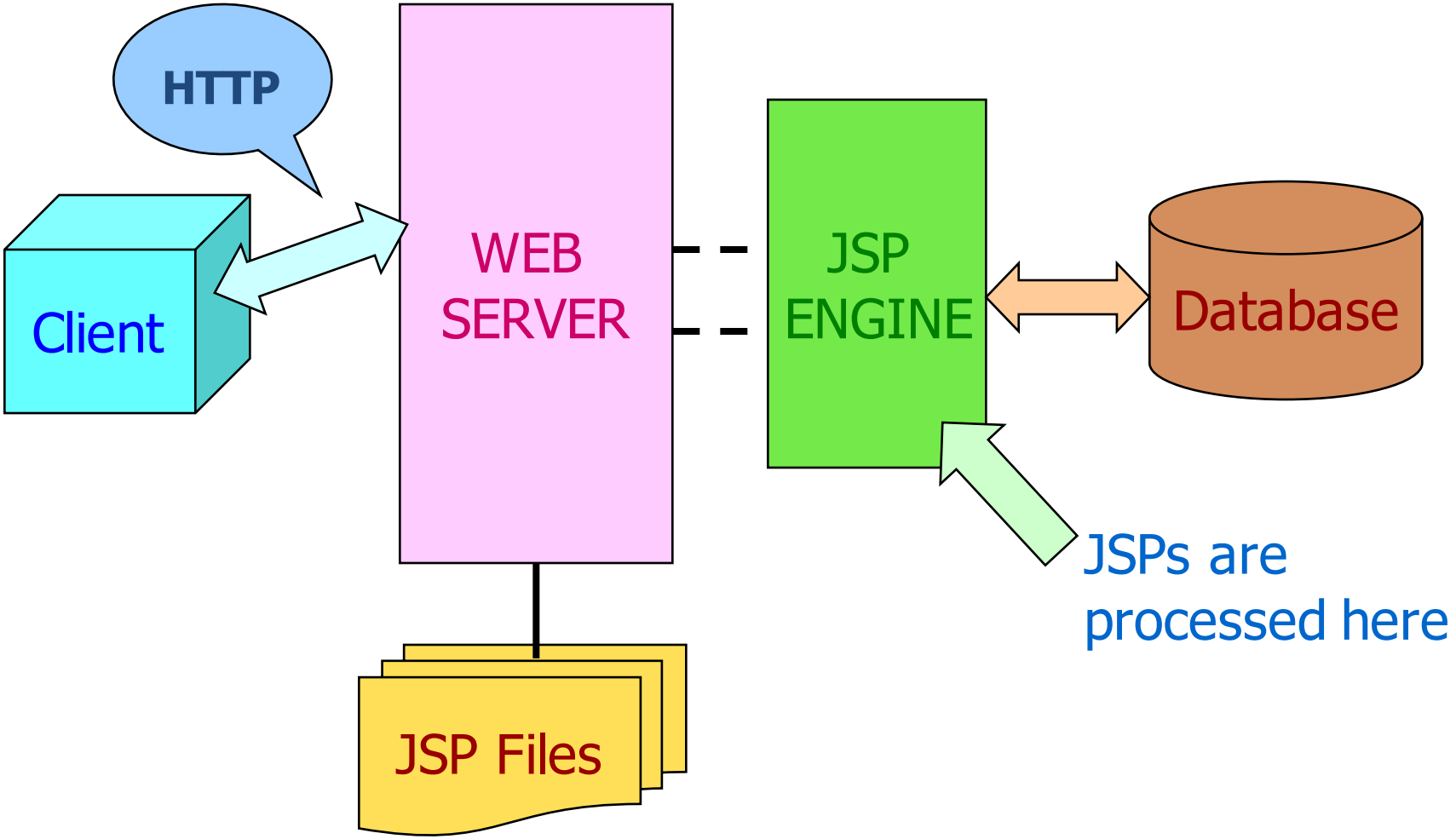
description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
org.apache.jasper.JasperException: Unable to compile class for JSP:

An error occurred at line: 15 in the jsp file: /index.jsp
Date cannot be resolved to a type
12:         <title>A simple date</title>
13:     </head>
14:     <body>
15:         The time on the server is <%= new Date()%>
16:     </body>
17: </html>
18:


Stacktrace:
```

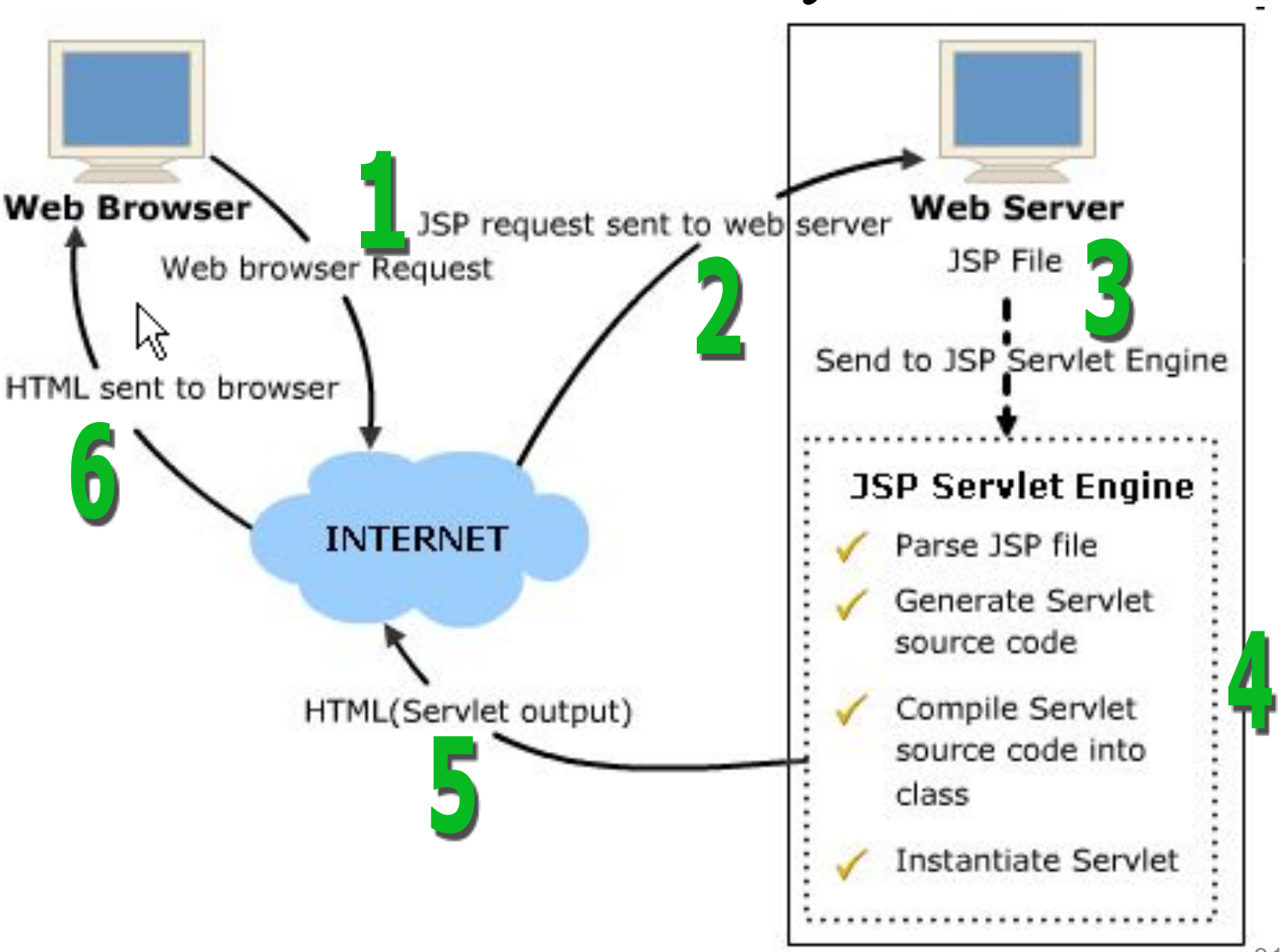
Java Server Pages

JSP Life Cycle



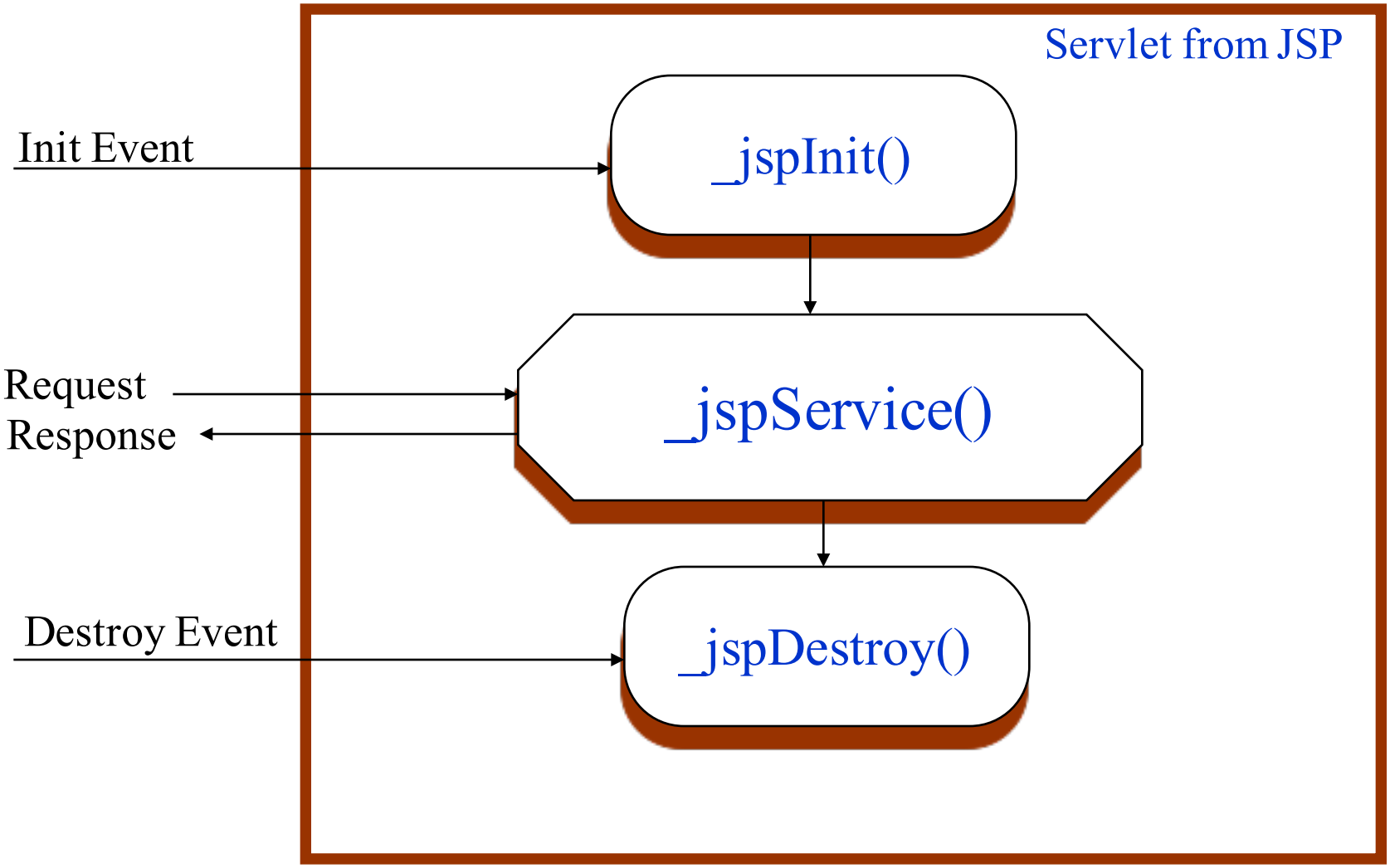
Java Server Pages

JSP Life Cycle



Java Server Pages

JSP Life Cycle



JSP Elements

JSP Tags

- Tags
  - Interface
  - Functional
  - Encapsulation
  - A tag starts with “<” and ends with “>”.
  - The tags contain body and attributes.
- The 4 types of JSP tags
  - Comment
  - Directives
  - Scripting elements
  - Standard actions

JSP Elements

Comments

- Are used for **documenting** JSP code
- Should **not be visible** to the client
- **Explains** the functioning of the code
- Comments **are ignored by the servlet** during compilation
- A JSP page contains **03 types** comments as JSP, HTML, and scripting
  - **HTML comments**
    - Are passed to resulting HTML documents
    - Are not visible in the output but the end user can view them.
  - **JSP comments**
    - The browser cannot view these comments as a part of the source code
    - Are ignored by the JSP to scriptlet translator.
  - **Scripting language comments**
    - Are written scriptlets in a JSP page.
    - The comment should be in the same comment syntax used for scripting language.
    - Are not visible in the output of the JSP page

JSP Elements
Comments

- Syntax
  - **JSP comments: <%-- comments --%>**
  - Ex: <%-- a JSP comment --%>
  - **HTML comments: <!-- comments -->**
  - Ex: <!-- a HTML comment -->
  - **Scripting language comment**
  - **<%/* comments */%>**
  - **<%// comments %>**
  - Ex: <%//It's is a variable declaration %>

JSP Elements

Scripting Elements

- A way of **performing server-side operations** in a JSP page
- Enable the **code to be directly embedded** in a JSP page
- **Insert** Java code into the JSP page
- **Declarations**:
  - Defines the **variables and methods** for a JSP page
  - Are **inserted** into the servlet, **outside** the **_jspservice**() method
  - Are used in **combination** with scriptlets and expressions to display an output.
  - A **single** declaration tag can be used to **define multiple variables**
  - **Syntax**: **<%! Declaration; %>**
  - **Ex**: <%! String s = “Aptech”; %>

JSP Elements

Scripting Elements

- **Scriptlets**
  - Is used to embed Java code, which is **inserted into the _jspService() method** of the servlet within an HTML code
  - Refers to **code blocks** executed for every request (a **fragment codes**)
  - Are used to add complex data to an HTML form
  - **Syntax**: **<% scriptlet %>**
  - **Ex**: <% for (int i =0 ; i<n; i++){
    System.out.println(i + ".This is scriptlets."); } %>
- **Expressions**
  - Can be used to **display individual variables or the result** of some calculation
  - Contains a Java statement whose value will **be evaluated** and **inserted** into the generated web page
  - Refers to **single line codes** executed for every request.
  - Provides **dynamic output generation and the result** is converted into a string
  - Evaluates at HTTP request
  - A declaration block is enclosed between delimiters.
  - **Syntax**: **<%= expression %>**
  - **Ex**: <%= i %>

JSP Directives

Directives

- **Controls the structure of the servlet** by sending messages from the JSP page to the JSP container.
- The **scope** of directives is the **entire JSP file**
- JSP uses directives for **controlling the processing** of JSP pages.
- **Do not produce any output** and **inform** the JSP **engine** about the **actions** to be performed on the JSP page
- **Specify scripting language used**
  - **Ex**: <%@ page language = "java" ...%>
- **Denote the use of custom tags library (taglib)**
  - **Ex**: <%@ taglib uri = "c:\..." prefix = "abc" %>
- **Include the contents of another JSP page into the current page**
  - **Ex**: <%@ include file = "c:\..." %>
- **Include Java file to Java packages list.**
  - **Ex**: <%@ page …. import = "java.util.*, java.lang.*" %>
- **Error handle** in JSP page and JSP page is catched errors (isErrorPage)
  - **Ex**: handle error <%@ page … errorpage = "/error.jsp" … %>

    process error <%@ page isErrorPage = "true" %>

Page Directives

- Is used **to define and manipulate** a number of important attributes that affect the entire JSP page
- Is written **at the beginning** of a JSP page
- A JSP page can contain any number of page directives. All directives in the page are processed together during translation and result is applied together to the JSP page
- **Syntax**: <%@ page **attributes** %>

Attributes	Descriptions
language	**Define** the **scripting language** used in the page. Default value is Java
extends	Change the content of the servlet that is generated
import	**Include** Java files to the Java package import list. Separating uses commas
session	**Specify** if the JSP page takes part in a HTTP session. Default value is true
buffer	**Specify** the size of the page **buffer**. Default value is 8KB
autoflush	**Flush** the page buffer is filled. Default value is true
isThreadSafe	**Define the safety level** of threads in the page. The JSP engine queues the requests sent for processing when the value is set to false. Default value is true
info	**Describe** the page
errorPage	**Define** the page to display **errors ocurring** in the JSP page
isErrorPage	**Indicate** the **current** JSP page if contains the path another error page of JSP
contentType	**Set** the **content type** or MIME type and character encoding for JSP. Default value is text/html

JSP Directives

Include & Tablib Directives

- **Include**
  - Is used to **physically include** the contents of another file sending to the server. The included file can be a HTML or JSP
  - Identify the file through the local URL
  - A single JSP file can include multiple include directives
  - **Syntax**: **<%@ include file = "URL" %>**
- **Taglib**
  - Enables the **use of custom tags** in the JSP page
  - Access to all the objects that are available for a JSP page
  - Extend the functionality of a JSP page one after the other
  - The TLD – Tag Library Descriptor is identified through the URI – Uniform Resource Identifier and prefix describes the prefix string used to define the custom tag
  - **Syntax**: **<%@ taglib uri = "URL" prefix = "name" %>**

Example

```
<%@ page import="java.util.Date" %>
<html>
  <head>
    <title>First JSP program</title>
  </head>
  <body>
    <!-- myFirstProgram.jsp -->
    <% out.println("Hello there!"); %><br>
    <%= "Current date is " + new Date() %>
    <%-- end Program --%>
</body>
</html>
```

Directives - page

HTML Comments

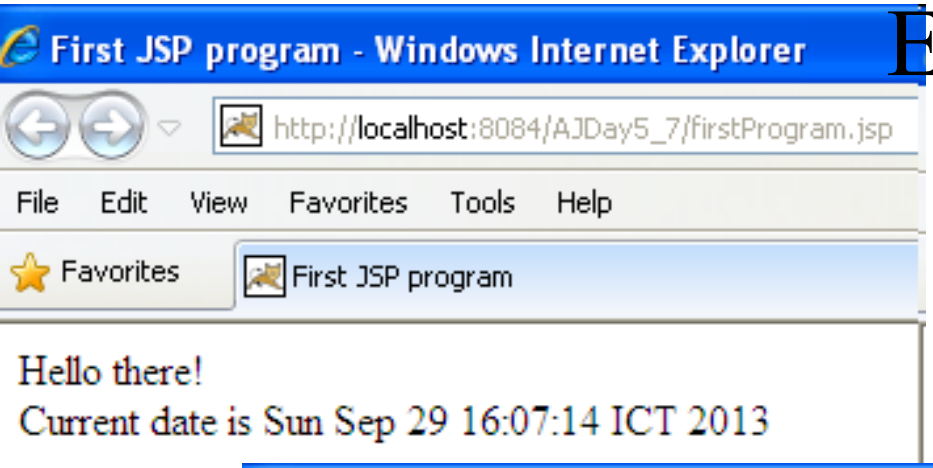
Scriptlet

Expression

JSP Comments

JSP Elements

Example



http://localhost:8084/AJDay5_7/firstProgram.jsp - Original Source

File Edit Format

```
1
2
3
4
5   <!DOCTYPE html>
6   <html>
7       <head>
8           <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
9           <title>First JSP program</title>
10    </head>
11    <body>
12          <!-- myFirstProgram.jsp -->
13          Hello there!
14  <br>
15          Current date is Sun Sep 29 16:07:40 ICT 2013
16
17      </body>
18  </html>
```

JSP Elements
Example

- **Main (main.jsp) file**

```
<html>
  <head>
    <title>Directive Includes JSP program</title>
  </head>
  <body>
    <%-- include use directives --%>
      Current date is
    <%@ include file = "myDate.jsp" %>
  </body>
</html>
```

JSP Comments

Directives - include

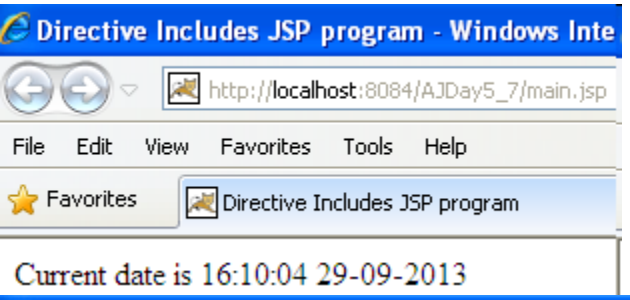
- **Include (myDate.jsp) file**

```
<%@ page import="java.util.Date" %>
<html>
  <head>
    <title>Date JSP program</title>
  </head>
  <body>
    <%= new Date().toLocaleString() %>
  </body>
</html>
```

Directives - page

Expression

JSP Elements Example



http://localhost:8084/AJDay5_7/main.jsp - Original Source

File Edit Format

C:\Documents and Settings\Trong Khanh\Application Data\NetBeans\7.2.1\apache-tomcat-7.0.27.0_base\work\Catalina\localhost\AJDay5_7\org\apache\jsp

main_jsp
CLASS File
5 KB

main_jsp
Java Source File
5 KB

```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Directive Includes JSP program</title>
  </head>
  <body>

        Current date is




<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Date JSP program</title>
  </head>
  <body>
      16:10:04 29-09-2013
    </body>
</html>

    </body>
</html>
```

JSP Elements

Example

main_jsp.java

Source | History

```
70          out.write("\n");
71          out.write("<!DOCTYPE html>\n");
72          out.write("<html>\n");
73          out.write("    <head>\n");
74          out.write("        <meta http-equiv=\"Content-Type\" content=\"text/html;
75          out.write("        <title>Directive Includes JSP program</title>\n");
76          out.write("  </head>\n");
77          out.write("  <body>\n");
78          out.write("  \t");
79          out.write("\n");
80          out.write("  \tCurrent date is\n");
81          out.write("  \t");
82          out.write("\n");
83          out.write("\n");
84          out.write("\n");
85          out.write("\n");
86          out.write("<!DOCTYPE html>\n");
87          out.write("<html>\n");
88          out.write("    <head>\n");
89          out.write("        <meta http-equiv=\"Content-Type\" content=\"text/html;
90          out.write("        <title>Date JSP program</title>\n");
91          out.write("  </head>\n");
92          out.write("  <body>\n");
93          out.write("      ");
94          out.print( new Date().toLocaleString() );
95          out.write("\n");
96          out.write("    </body>\n");
97          out.write("</html>\n");
```

JSP Elements Example

main.jsp

Source | History

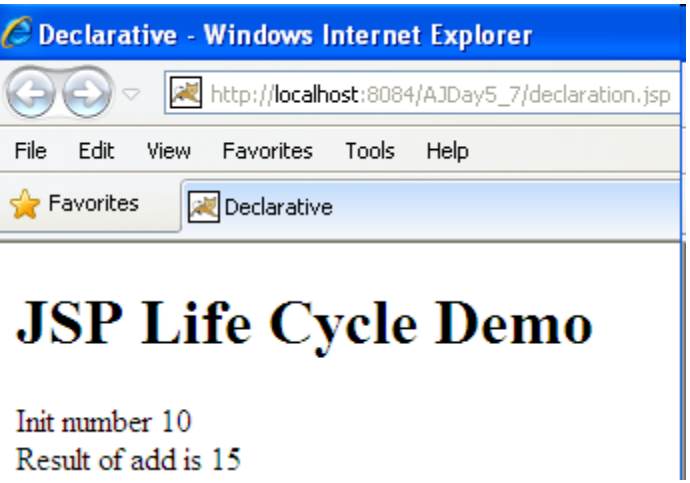
```
<html>
    <head>
        <title>JSP life</title>
    </head>
    <body>
        <h1>JSP Life Cycle demo</h1>
        <%!
            int num;

            public void jspInit() {
                System.out.println("JSPInit is invoked!!!");
                num = 10;
            }

            public void jspDestroy() {
                System.out.println("JSPDestroy is invoked!!!!");
                num = 0;
            }

            public int add(int n) {
                System.out.println("Add is called!!");
                num += n;
                return num;
            }

        %>
        Init number <%= num %><br/>
        Result= <%= add(5)%>
    </body>
</html>
```

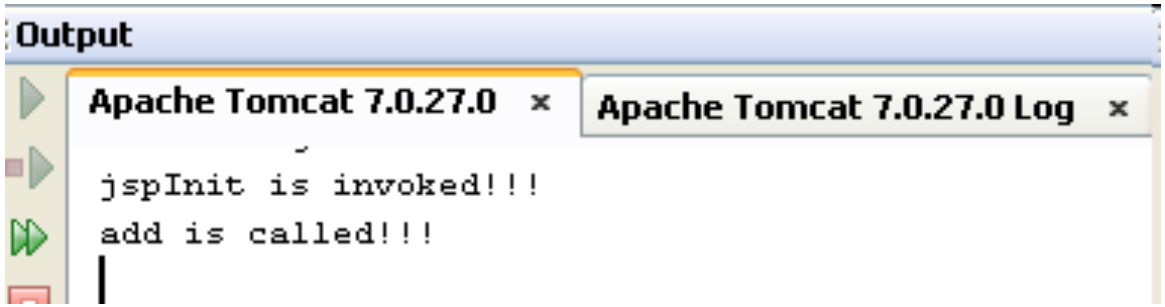
jspInit

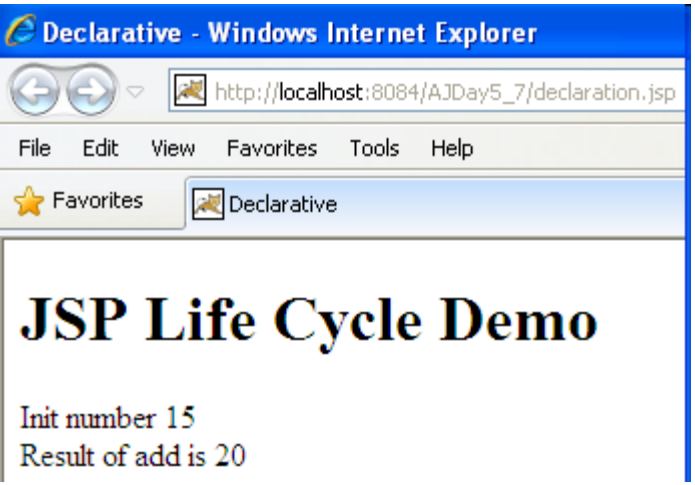
Declarations

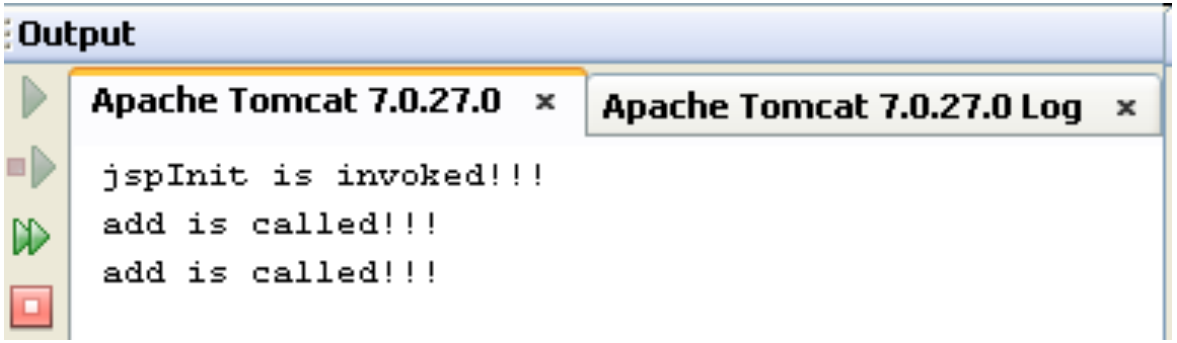
jspDestroy

JSP Elements Example



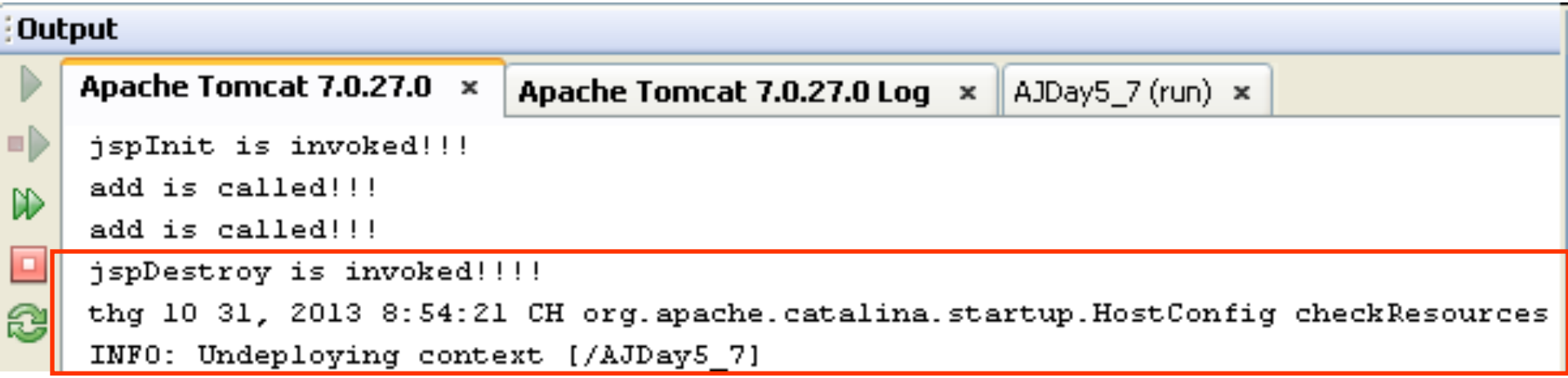


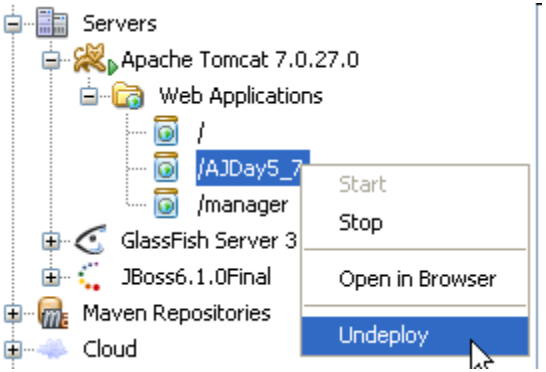


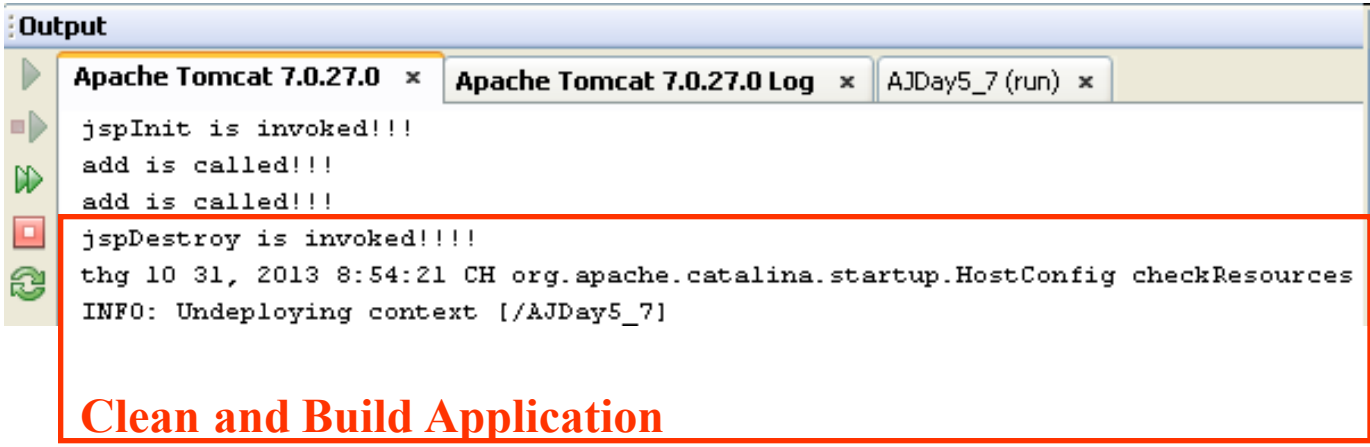


JSP Elements

Example







JSP Elements Example

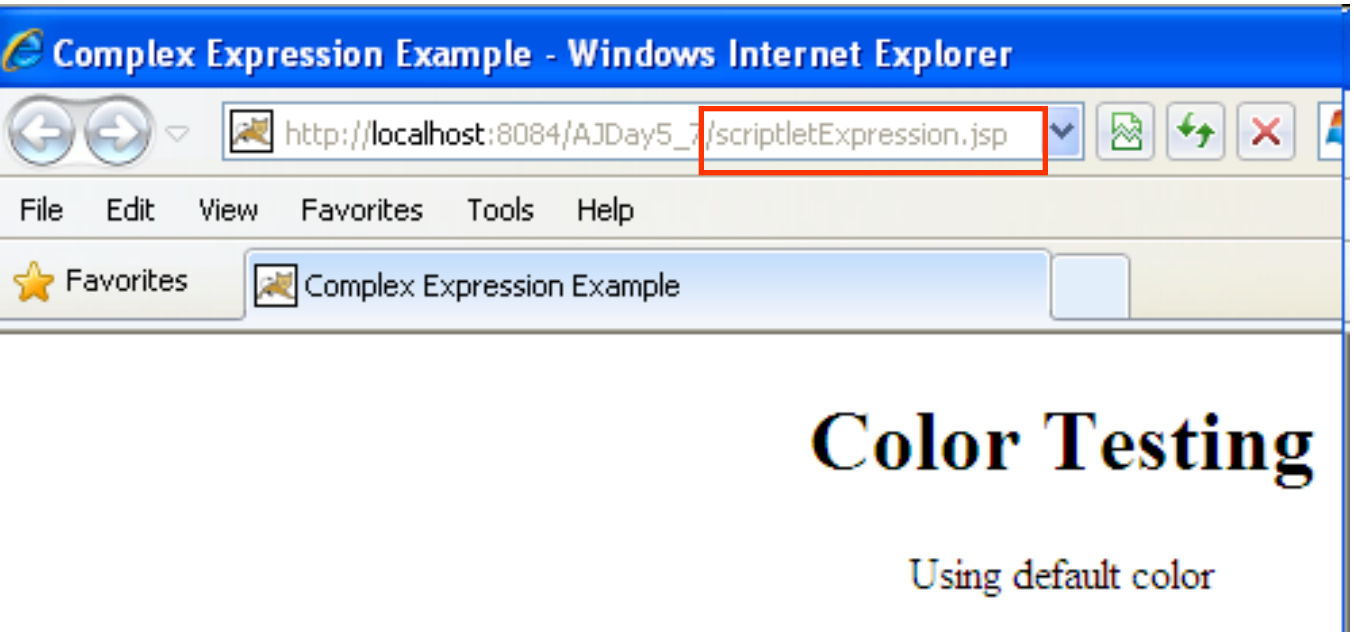
scriptletExpression.jsp

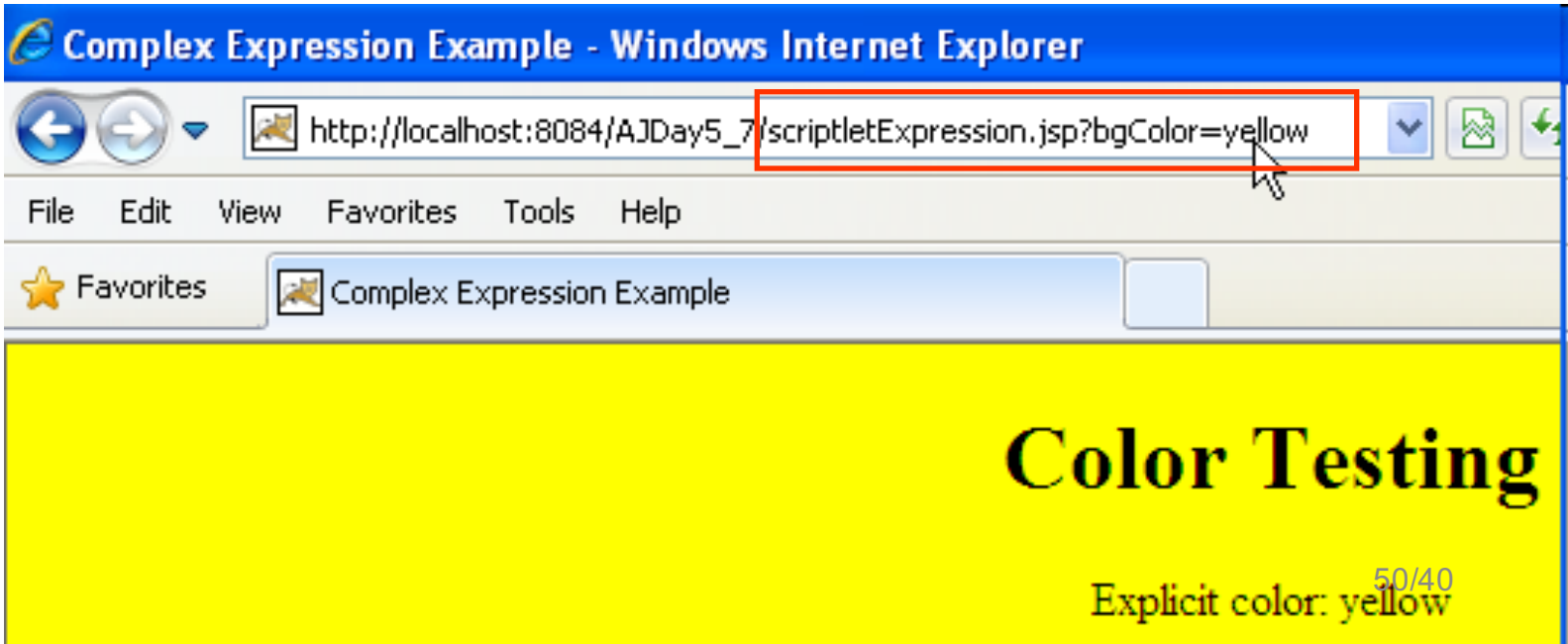
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        ><title>Complex Expression Example</title></head>
        <%
                    String bgColor = request.getParameter("bgColor");
                    boolean hasExplicitColor;
                    if (bgColor != null) {
                        hasExplicitColor = true;
                    } else {
                        hasExplicitColor = false;
                        bgColor = "white";
                    }
        %>
    <body bgcolor="<%= bgColor%>">
        <center>
            <h1>Color Testing</h1>
            <%
                        if (hasExplicitColor) {
                            out.println("Explicit color: " + bgColor);
                        } else {
                            out.println("Using default color");
                        }
            %>
        </center>
    </body>
</html>
```

Expression!!!

- Run jsp page without parameters
- Run with parameters as **yellow, blue, red ...**

JSP Elements Example





JSP Elements

Example – Exception

testDemo.jsp

```
14          <title>Scripting</title>
15      </head>
16      <body>
17          <h1>Scripting Demo</h1>
18          <%!
19              String s = "declaration";
20
21              public void test() {
22                   System.out.println("This is a test function!!!");
23              }
24          %>
25
26          <%
27               String s = "scriptlet";
28
                 public void test1() {
30                   System.out.println("This is a function in scriptlet");
31               }
32          %>
33      </body>
34  </html>
```

```
Z:\LapTrinh\Servlet\AJ\AJDay5_7\build\generated\src\org\apache\jsp\testDemo_jsp.java:68: error: illegal start of expression
            public void test1() {
            ^
Z:\LapTrinh\Servlet\AJ\AJDay5_7\build\generated\src\org\apache\jsp\testDemo_jsp.java:68: error: illegal start of expression
            public void test1() {
                   ^
Z:\LapTrinh\Servlet\AJ\AJDay5_7\build\generated\src\org\apache\jsp\testDemo_jsp.java:68: error: ';' expected
            public void test1() {
                              ^
```

JSP Elements

Example – Exception

```jsp
<body>
    <h1>Scripting Demo</h1>
    <%!
        String s = "declaration";

        public void test() {
            System.out.println("This is a test function!!!");
        }
    %>

    <%
        String s = "scriptlet";
    %>
</body>
```

testDemo_jsp.java

```
 7  public final class testDemo_jsp extends org.apache.jasper.runtime.HttpJspBase
 8      implements org.apache.jasper.runtime.JspSourceDependent {
 9              String s = "declaration";
10
11              public void test() {
12                  System.out.println("This is a test function!!!");
13              }
14
15      private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();
```

```
35      public void _jspService(HttpServletRequest request, HttpServletResponse response)
36              throws java.io.IOException, ServletException {
```

```
69              out.write("    <body>\n");
70              out.write("        <h1>Scripting Demo</h1>\n");
71              out.write("        ");
72              out.write("\n");
73              out.write("\n");
74              out.write("        ");
75
                    String s = "scriptlet";
77
78              out.write("\n");
79              out.write("    </body>\n");
80              out.write("</html>\n");
81          } catch (Throwable t) {
```

```jsp
<%= s %> . out.print( s );
```

JSP Implicit Objects

Implicit Objects

- **Does not initialize or declare**
- **Are loaded** by the Web Container **automatically and maintains** them in a JSP page (**Available for scriptlets or expressions**)
- Created using directives and accessible according to the **specified scopes**
- The names of the implicit objects **are reserved words** of JSP
- The scopes for the IB in JSP page including **page, request, session, and application**
- **Access dynamic content** using JavaBeans
- **Syntax**: **ImplicitObject.method(params)**
- **Types** of Implicit Objects
  - ***Input & Output Objects***
    - The objects control page input and output
    - Are request, response and out
  - ***Scope Communication Objects***: provide access to all objects available in the given scope
  - ***Servlet Objects***
    - Provides information about the page context
    - Processes request objects from a client and sends the response objects back to the client

JSP Implicit Objects

- Types of Implicit Objects (cont)
  - The Error Objects
    - The object handles errors in a JSP page (exception)
    - Can access this object by declaring your JSP page as an error page <%@page isErrorPage="true" %>

Object	Class / Interface
page	javax.servlet.jsp.HttpJspPage – variable synonym for this object
config	javax.servlet.ServletConfig
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
pageContext	javax.servlet.jsp.PageContext
exception	java.lang.Throwable

JSP Implicit Objects
Input & Output Objects

Objects	Descriptions
request	- Refer to the current request made by the client that is being processed by JSP container. The container passed the request IB to JSP page as a parameter to the _jspservice(). - Implement the **javax.servlet.http.HttpServletRequest** interface - **Syntax**: request.method(params) - **Scope**: request -**Ex**: request.getParameter("username");
response	- Refers the result that is returned to the user after a JSP processed - Implement the **javax.servlet.http.HttpServletResponse** interface - **Syntax**: response.method(params) - **Scope**: page - **Ex** : response.addCookie(cookie)
out	- Represent the output stream for the JSP page (send to client) - Implement the **javax.servlet.jsp.JspWriter** interface (the buffer size is supported by Servlet) - **Syntax**: out.method(params) - **Scope**: page - **Ex**: out.println("output stream")

JSP Implicit Objects

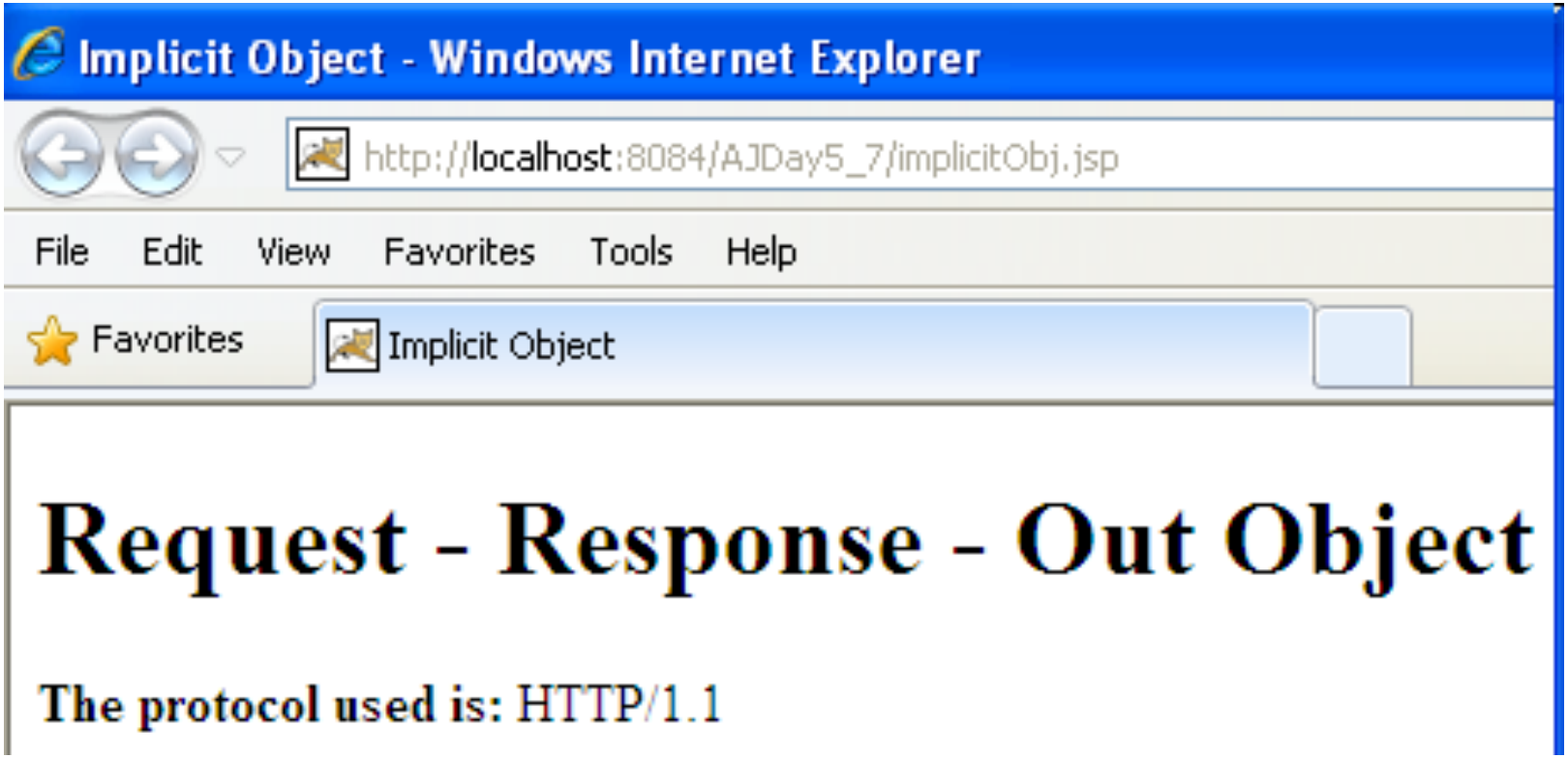
Input & Output Objects – Example

implicitObj.jsp

```
1  <%--...--%>
6
7  <%@page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
9     "http://www.w3.org/TR/html4/loose.dtd">
10
11 <html>
12     <head>
13         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14         <title>Implicit Object</title>
15     </head>
16     <body>
17
18     <h1>Request - Response - Out Object</h1>
19     <%
20         response.setDateHeader ("Expires", 0);
21         response.setHeader ("Pragma", "no-cache");
22         if(request.getProtocol ().equals ("HTTP/1.1")){
23             response.setHeader ("Cache-Control", "no-cache");
24             out.println ("<b>The protocol used is:</b> " + request.getProtocol ());
25         }
26     %>
27     </body>
28 </html>
```

JSP Implicit Objects

Input & Output Objects – Example



JSP Implicit Objects

Scope Communication Objects

Objects	Descriptions
session	- Specify data and store information in the current session. - Implement the **javax.servlet.http.HttpSession** interface - **Syntax**: session.method(params) - **Scope**: session - **Ex**: session.setAttribute("username", "Aptech");
application	- Represent the application of the required JSP page and represent the servlet context about Web Application in which it is running. - Implement the **javax.servlet.ServletContext** interface - **Syntax**: application.method(params) - **Scope**: application - **Ex**: application.setAttribute("username", "Aptech");
pageContext	- An instance of Pages (**javax.jsp.PageContext**) - Enable access to the JSP page and the attributes associated with that page - provides following fields to find the scope or specify the scope of the objects (PAGE, REQUEST, SESSION, and APPLICATION) - **Syntax**: pageContext.method(params) - **Ex**: pageContext.getAttributes("username");

JSP Implicit Objects

Scope Communication Objects – Example

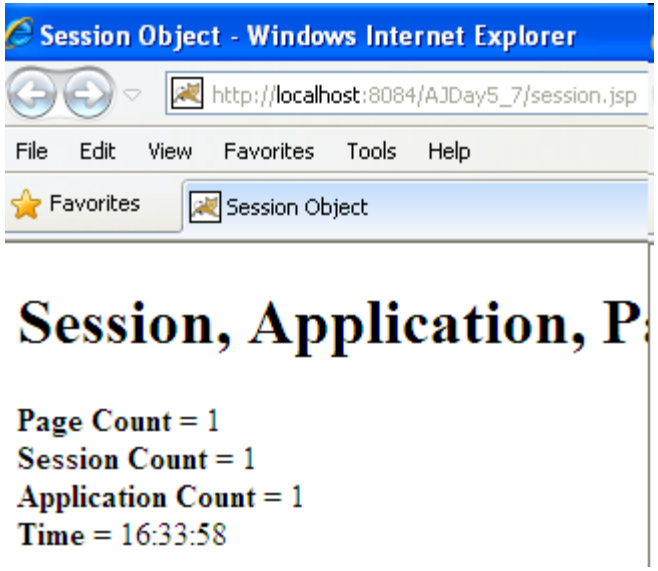
session.jsp

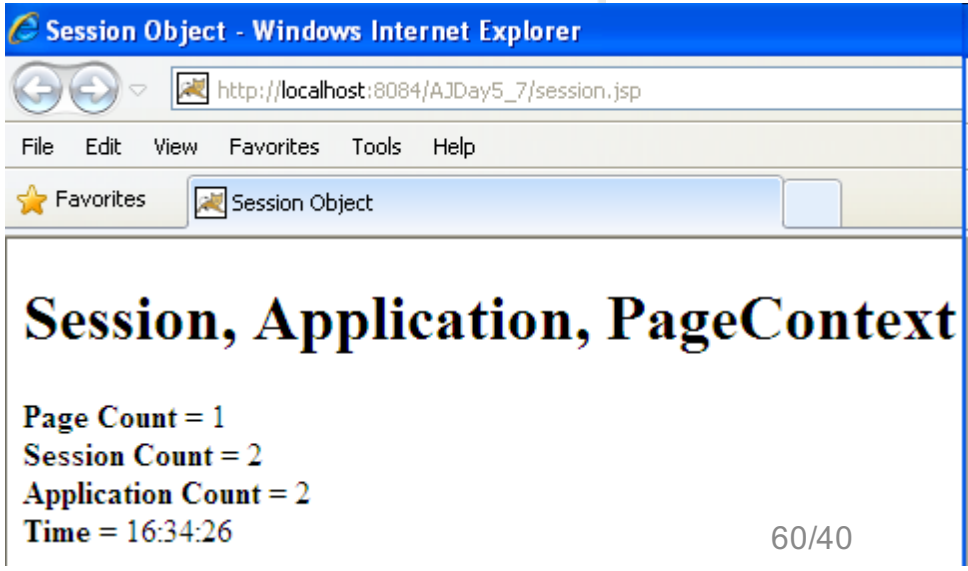
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<%
            if (pageContext.getAttribute("pageCount") == null) {
                pageContext.setAttribute("pageCount", new Integer(0));
            }
            if (session.getAttribute("sessionCount") == null) {
                session.setAttribute("sessionCount", new Integer(0));
            }
            if (application.getAttribute("appCount") == null) {
                application.setAttribute("appCount", new Integer(0));
            }
%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Session Object</title>
    </head>
    <body>
        <h1>Session, Application, PageContext</h1>
        <%
                    Integer count = (Integer) pageContext.getAttribute("pageCount");
                    pageContext.setAttribute("pageCount", new Integer(count.intValue() + 1));
                    Integer count2 = (Integer) session.getAttribute("sessionCount");
                    session.setAttribute("sessionCount", new Integer(count2.intValue() + 1));
                    Integer count3 = (Integer) application.getAttribute("appCount");
                    application.setAttribute("appCount", new Integer(count3.intValue() + 1));
        %>
```

JSP Implicit Objects

Scope Communication Objects – Example

```
35          <b>Page Count =</b>
36          <%= pageContext.getAttribute("pageCount")%><br/>
37
38          <b>Session Count =</b>
39          <%= session.getAttribute("sessionCount")%><br/>
40
41          <b>Application Count =</b>
42          <%= application.getAttribute("appCount")%><br/>
43
44          <b>Time =</b>
45          <%= new java.sql.Time(System.currentTimeMillis())%><br/>
46      </body>
47  </html>
```





JSP Implicit Objects
Servlet Objects

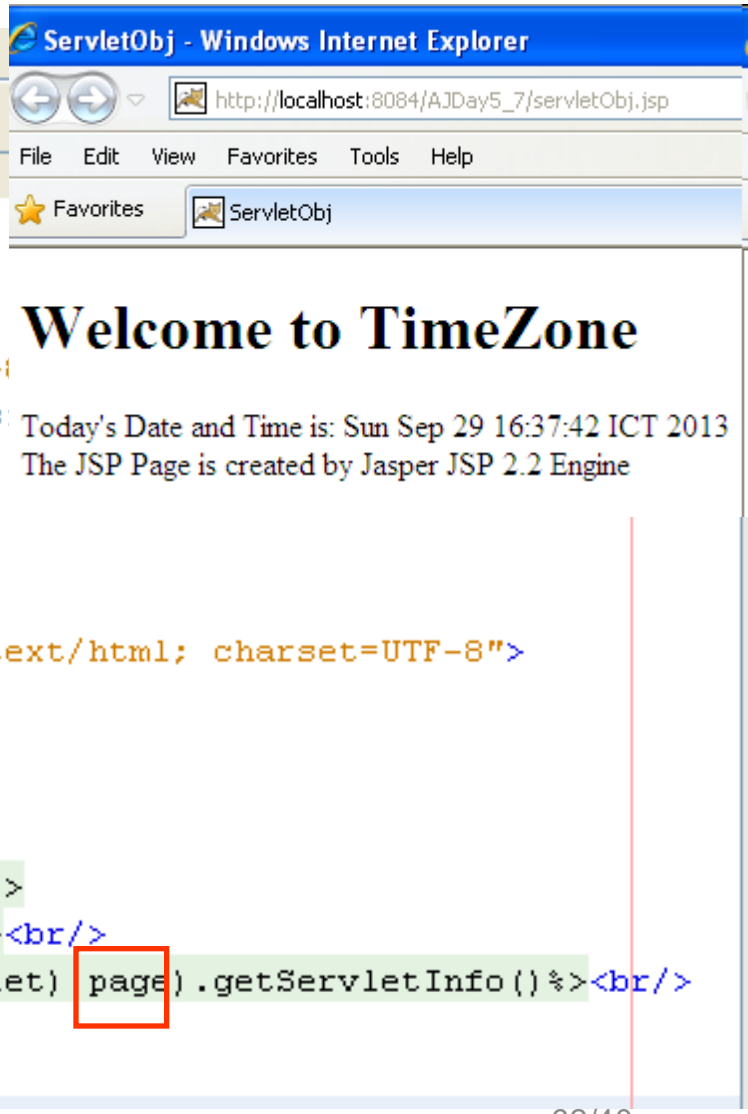
Objects	Descriptions
page	- Represents the servlets and the initialization parameters of the servlet are stored in the config IB - Use "this" reference and the page IB represents it. - Implements the **javax.lang.object** interface - **Syntax**: **<%@ page info = "information" %>** - **Scope**: page
config	- Represent the configuration of the servlet data - Implement the **javax.Servlet.ServletConfig** interface - Access objects through config.getInitParameter("par") - **Scope**: page

JSP Implicit Objects

Servlet Objects – Example

servletObj.jsp

```
6
7   <%@page import="java.util.Date"%>
8   <%@page import="java.util.Calendar"%>
9   <%@page contentType="text/html" pageEncoding="UTF-
10  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Trans
11      "http://www.w3.org/TR/html4/loose.dtd">
12
13  <html>
14      <head>
15          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
16          <title>ServletObj</title>
17      </head>
18      <body>
19          <h1>Welcome to TimeZone</h1>
20          <% Calendar cal = Calendar.getInstance();%>
21          Today's Date and Time is: <%= new Date()%><br/>
22          The JSP Page is created by <%= ((HttpServlet) page).getServletInfo()%><br/>
23      </body>
24  </html>
```



JSP Implicit Objects

Error Objects

Objects	Descriptions
exception	- Refer to the runtime exception in an error page - Is available only on pages that are assigned as error page using the **isErrorPage** attribute of the page directive. - Implement the **javax.lang.Throwable** interface - **Exception** methods are supported + **getMessage**() : return the error message associated with the exception + **toString**() : Return a string with the class name of the exception within the error message. + **printStackTrace**(): prints the execution stack in effect when the exception was thrown to the designated output stream

JSP Implicit Objects

Error Objects – Example

- A JSP page traps errors occurred

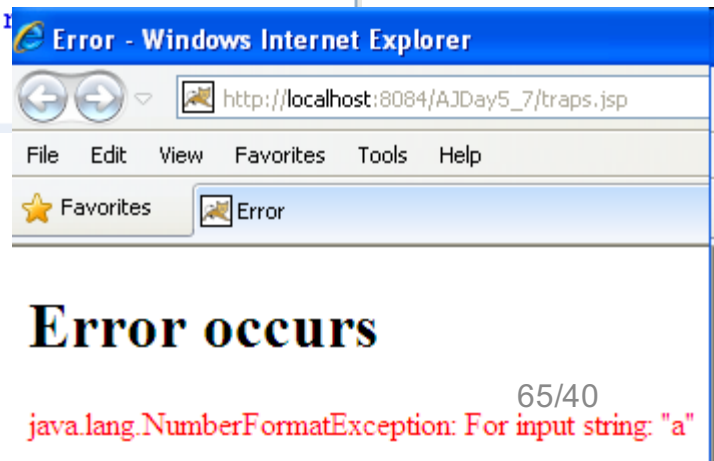
```
<%@ page errorPage="error.jsp" %>
<html>
  <head> <title>Tag Example</title> </head>
  <body>
    <%! String name; %>
    <%          name = request.getParameter("name");
          if(name==null) name="World“; %>
    <h1>Hello, <%= name %></h1>
    <% out.println("<H1>Hello " + name + "</H1>"); %>
    <% int num=Integer.parseInt("a"); %> </body>
</html>
```

JSP Implicit Objects

Error Objects – Example

error.jsp

```
1  <%--...--%>
6
7  <%@page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
9     "http://www.w3.org/TR/html4/loose.dtd">
10 <%@page isErrorPage="true" %>
11 <html>
12     <head>
13         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
14         <title>Error</title>
15     </head>
16     <body>
17         <h1>Error occurs</h1>
18         <font color="red"><%= exception.toString() %></fo
19     </body>
20 </html>
```



Summary

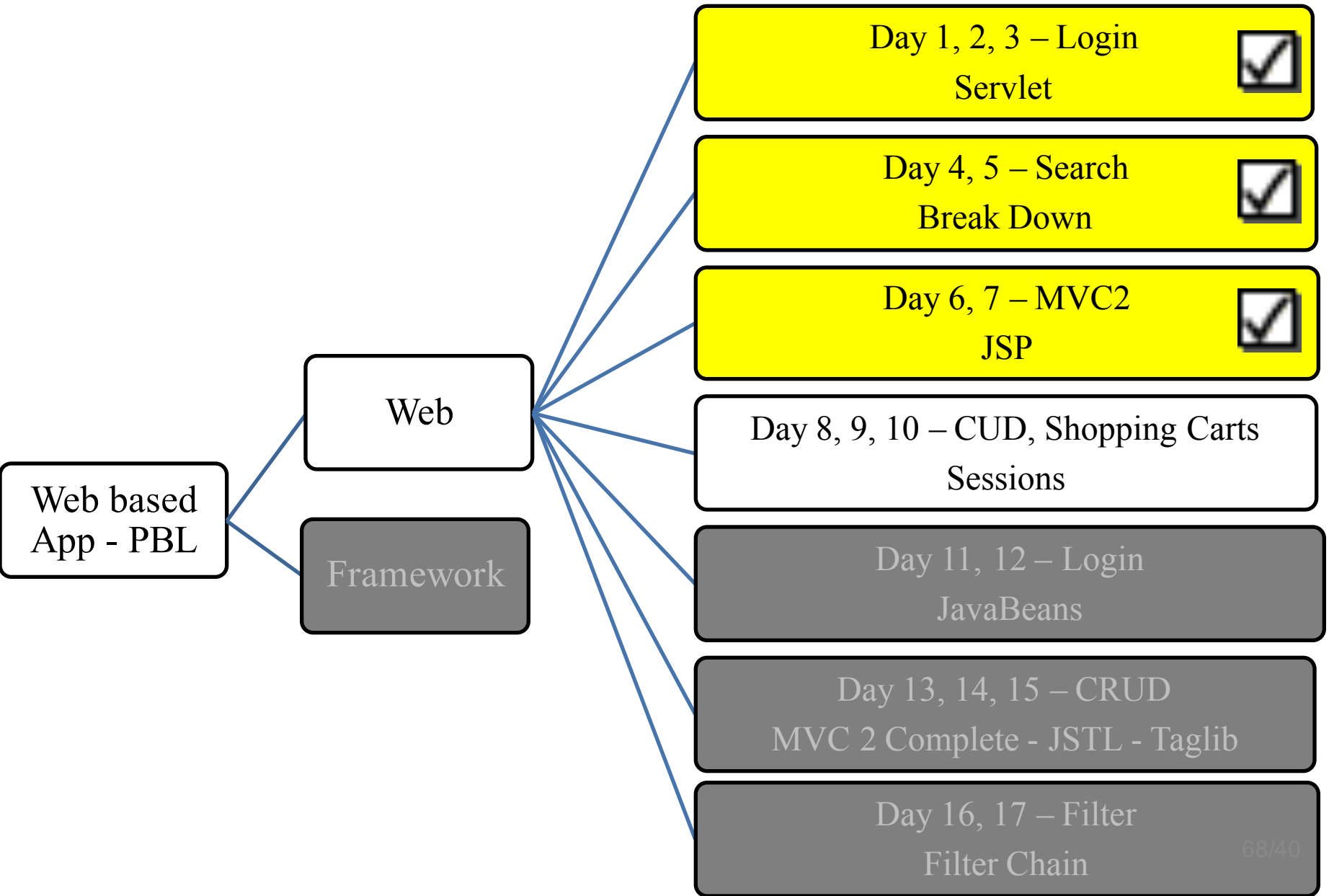
- **How to build web application applying MVC model using Servlet, JSP + Scripting Element**
  - JSP vs. Servlet
  - JSP mechanism, syntax
  - MVC Model
  - How to use JSP combining the Servlets and Java objects
  - How to connect DB using Dynamic connection or DataSource

Q&A

Next Lecture

- **How to write CUD Web Application**
  - Session Tracking Techniques
  - Manipulate DB Techniques in Web Application
  - Break down structure component in building web application
- **Techniques: Error Handling in Servlets**
  - Reporting Errors
  - Logging Errors
  - Users Errors vs. System Errors

Next Lecture



Appendix – MVC Design Pattern

Model – View – Controller

- **Main concern** of MVC is to **separate** the **data** (**model**) and the **user interface** (**view**)
  - Helps the developer **changing** to the **user interface** can be made **without affecting** the **underlying data** handling logic.
  - The data can be **reorganized without changing** the user interface.
- **Separation** is **achieved** by introducing an **controller** component
  - **Controller** is an **intermediate component**
  - Controller **defines** as how the **user interface** should **react** to a **user input**

Appendix – MVC Design Pattern

Model – View – Controller

- **Model**
  - Contains only the pure application data (it **contains no logic** describing how to **present** the **data to** a **user**)
  - Models **data** and **behavior behind** business process
  - Manages information (**access, modify, and represent application's data**) and notifies observers whenever the information changes
  - Maps Real-World Entities (**implement business logic, workflow**)
  - **Performing** DB **Queries**
  - **Calculating** Business Process
  - **Encapsulates Domain Logic** (a Java Object – **Java Bean**) which are independent of Presentation
- **View**
  - Obtains data from model & presents the model's data to the user
  - **Represents** Output/Input of the application **(GUI – JSP & HTML …)**
  - **Display** results of Business Logic
  - Free Access to Model, but should not change the state of the model.
  - **Reads** Data from Model – Using Query Methods

Appendix – MVC Design Pattern

Model – View – Controller

- **Controller**
  - Serves **logical connection between user's** interaction and the **business process**
  - It **receives** and **translates** input to request on model or view
  - **Input from user** and instructs the model and view to perform action
  - **Responsible** for making decision among multiple presentation
  - **Maps** the end-user action to the application response
  - Is **responsible** for **calling** methods on the model that changes the state of the model
  - **Updates the state of the Model and generates one or more views (servlets)**
- **Evolution** of MVC Architecture
  - NO MVC
  - MVC Model 1 [**Page-centric**] – JSP Model 1
  - MVC Model 2 [**Servlet-centric**] – JSP Model 2

Appendix – MVC Design Pattern

Model – View – Controller

- **Relationships** between components
  - **View** and **Controller**: Controller is responsible for creating or selecting view
  - **Model** and **View**
    - View depends on Model
    - If a change is made to the model then there might be required to make parallel changes in the view
  - **Model** and **Controller**
    - Controller depends on model
    - If a change is made to the model then there might be required to make parallel changes in the Controller
- **Logical Layers** in Web application
  - **Model** [Business Process Layer]
  - **View** [Presentation Layer]
  - **Controller** [Control Layer]

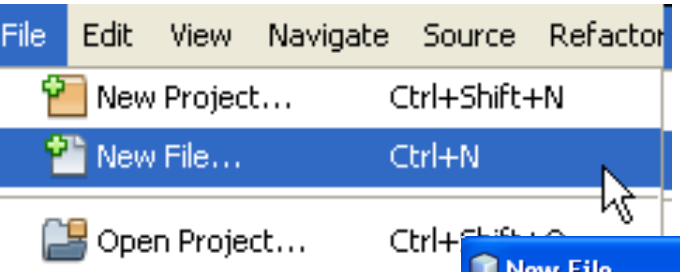
Appendix – MVC Design Pattern

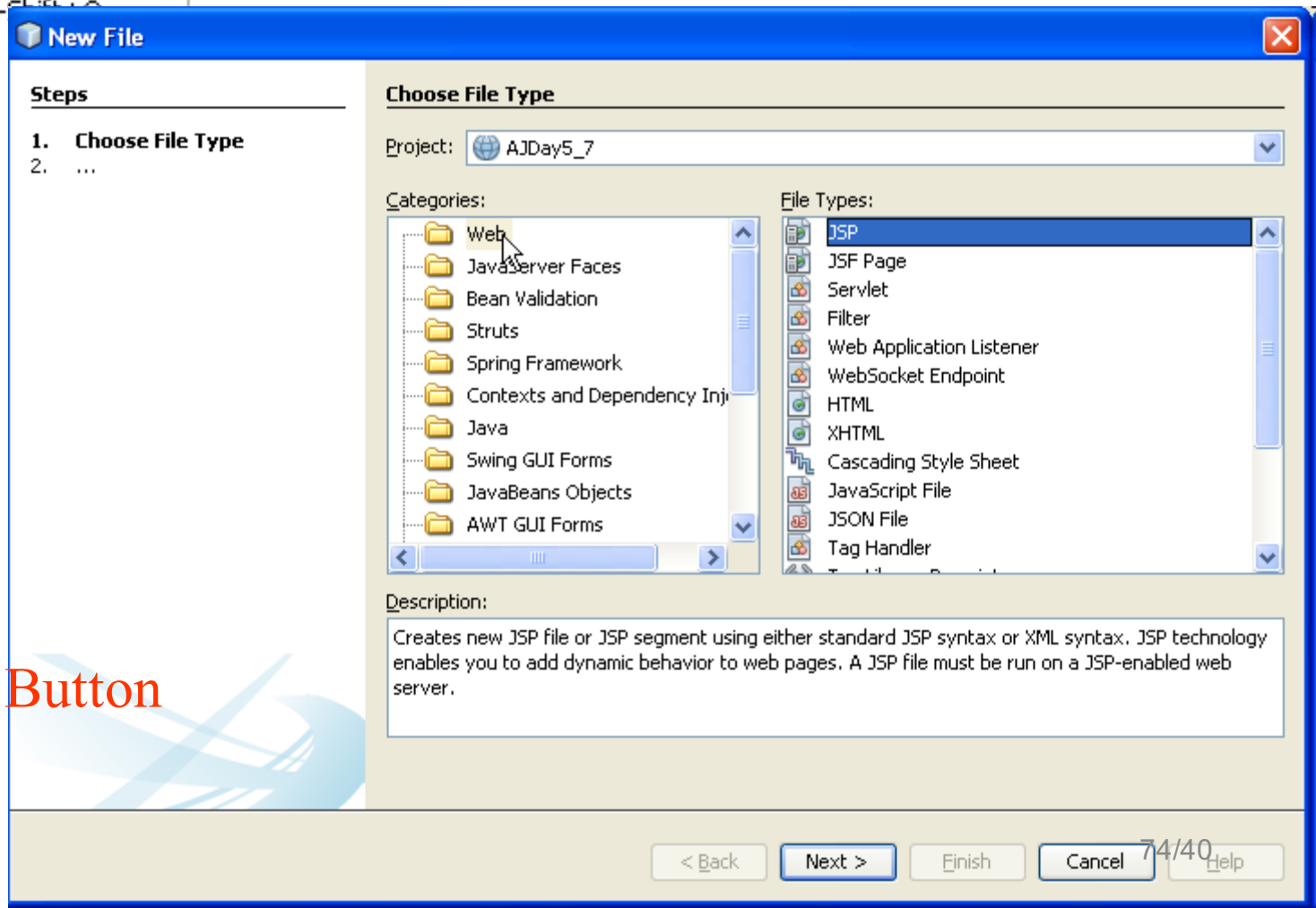
MVC Model 1

- Composed of a **series of interrelated JSP pages**
- **JSP** page **handles** the entire request processing mechanism
  - Extract the HTTP request parameters
  - Invoke the business logic (through Java Beans)
  - Process the business logic
  - Handle the HTTP session
- JSP page **responsible** for **displaying** the **output** to the client
  - There is no extra Servlet involved in the process.
- **A page centric architecture (Business process logic and control decisions are hard coded inside JSP pages)**
- Next page selection is determined by **hyperlink** or **action** of submitting a form. **Ex**:
  - Search
  - <form action="find.jsp"> … </form>

Appendix

Create the JSP Page on NetBeans

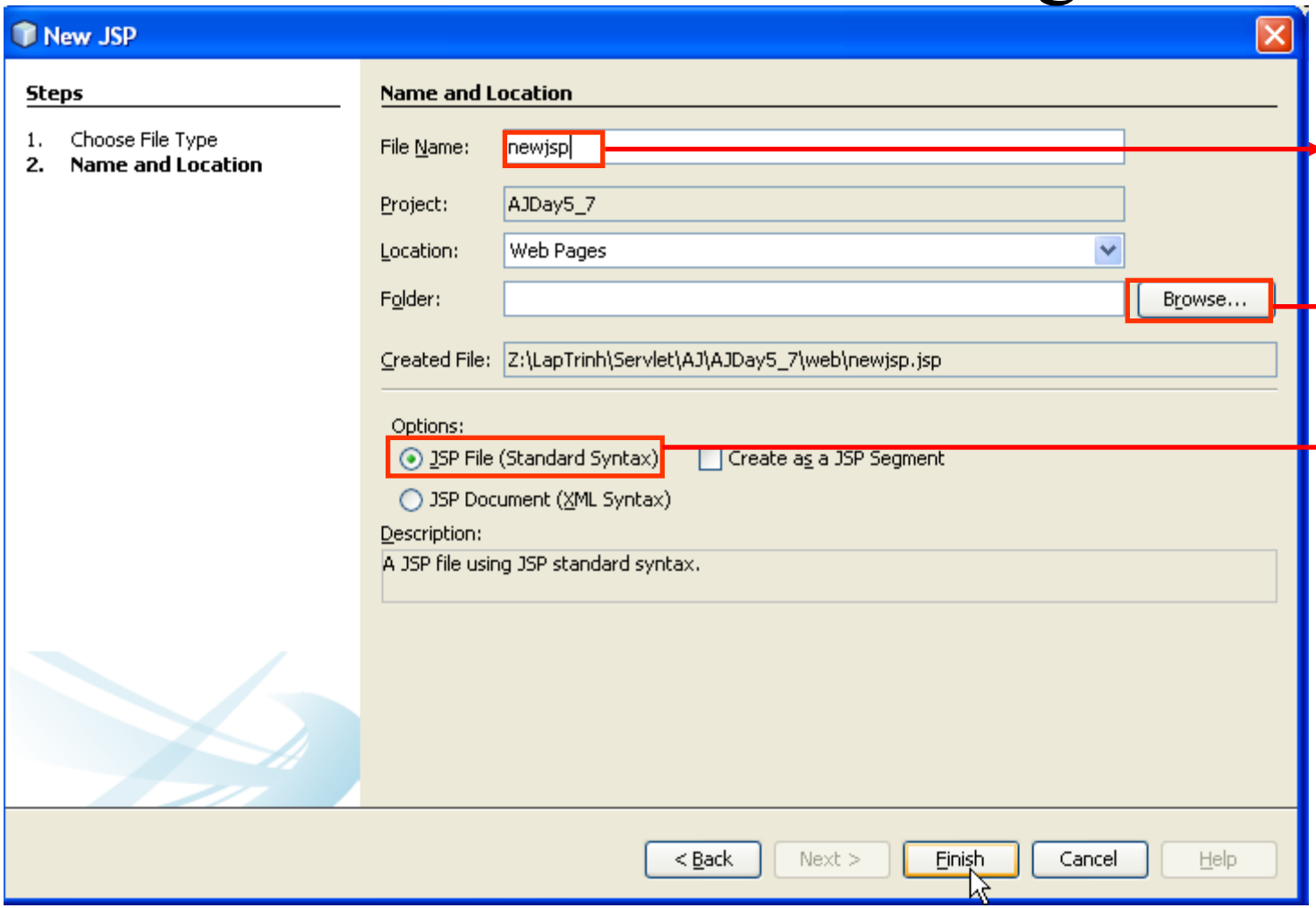




- Click **Next** Button

Appendix

Create the JSP Page on NetBeans



Fill your file name without .jsp

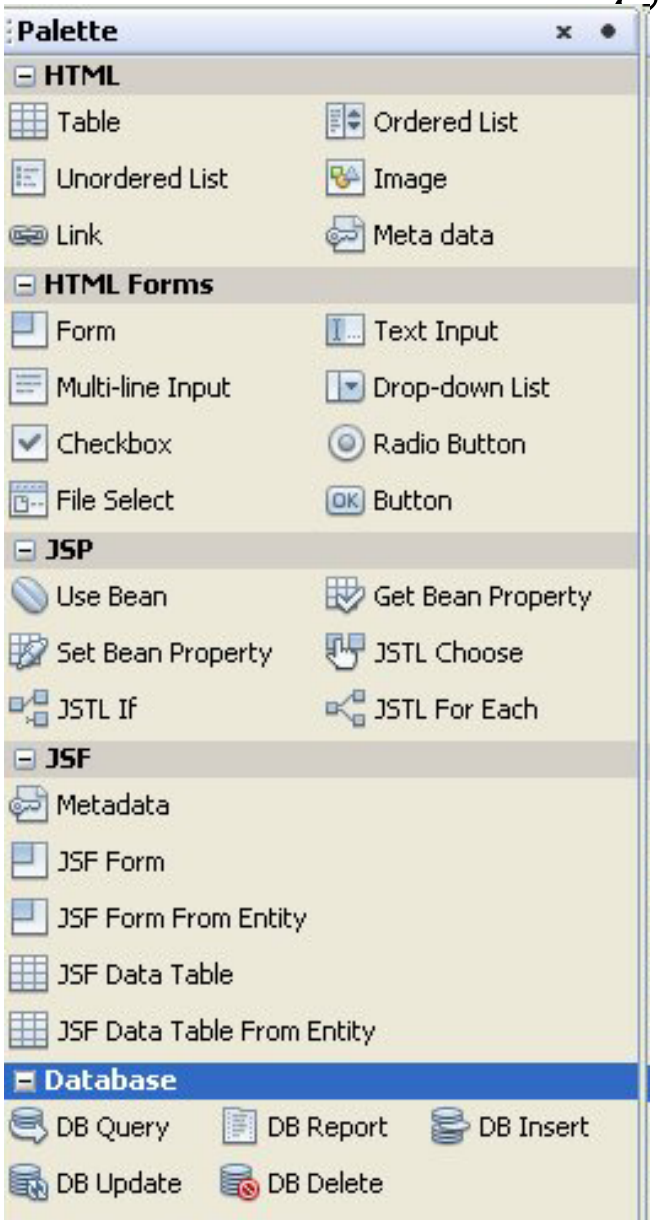
Locate the directory store jsp file in current project

Using JSP File

- Click **Finish** Button

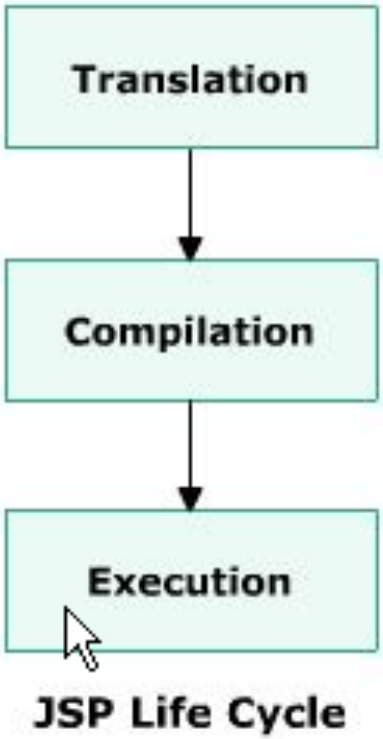
Appendix

Create the JSP Page on NetBeans



Java Server Pages

JSP Life Cycle

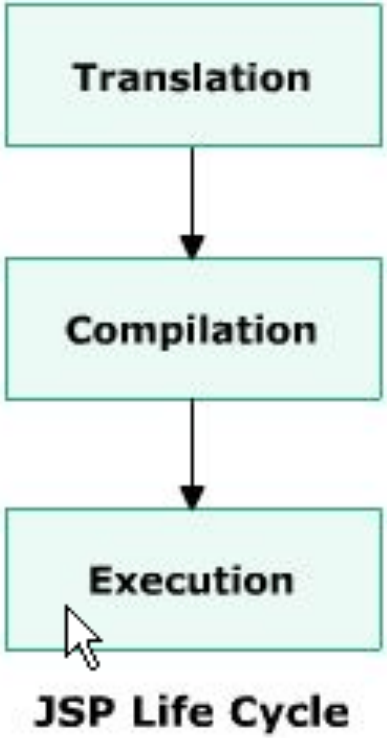


JSP Life Cycle

- **Translation**
  - A servlet code to implement JSP tags is automatically generated, complied and loaded into the servlet container.
  - **jspInit**() method is invoked when the JSP page is **initialized** and **requested**
  - The **_jspService**() method corresponds to the body of the JSP page and is **defined automatically** by the JSP container and **never** be **defined** by the JSP page
  - The **jspDestroy**() method is invoke when the JSP page is going to be **destroyed** (**requested again**)
  - **Notes**: the servlet must implement the **javax.servlet.jsp.HttpJspPage interface**

Java Server Pages

JSP Life Cycle



JSP Life Cycle

- **Complication**
  - The JSP page is automatically compiled and executed again by JSP/ Servlet Engine
- **Execution**
  - Is carried out with the help of page directives controlling various execution parameters and are used for buffering output and handling errors

JSP Elements

Overview

- Enables to **create dynamic JSP pages**
- The JSP server **translates** and **executes** JSP elements

Elements	Description
Root	Classifies standard elements and attributes of namespaces in tag library
Comment	Used in JSP file documentation
Declaration	Declares variables and methods in a scripting language page.
Expression	Includes expression in a scripting language page
Scriptlet	Includes code fragment in a scripting language page
Text	Includes data and text
include Directive	Includes content of one JSP page into the current JSP page

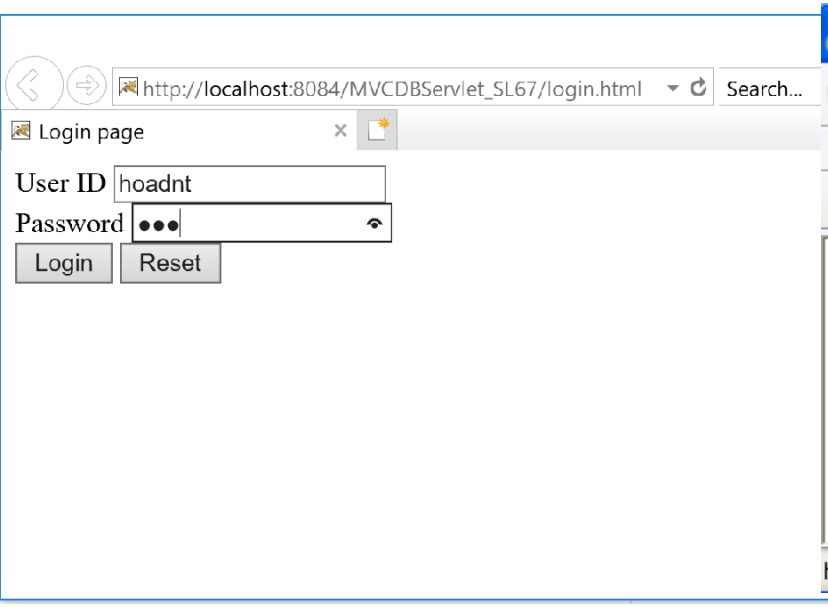
JSP Elements

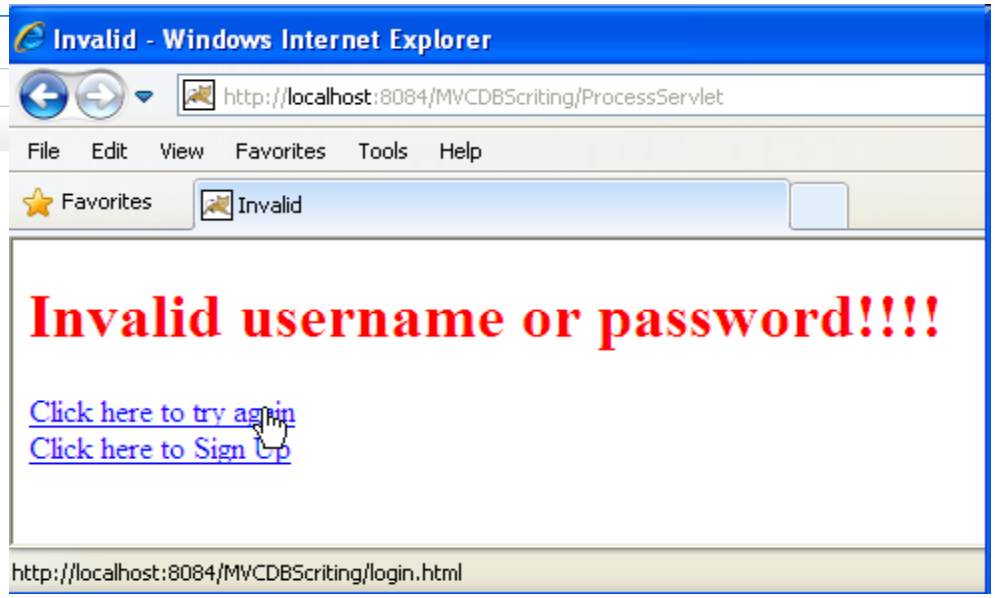
Overview

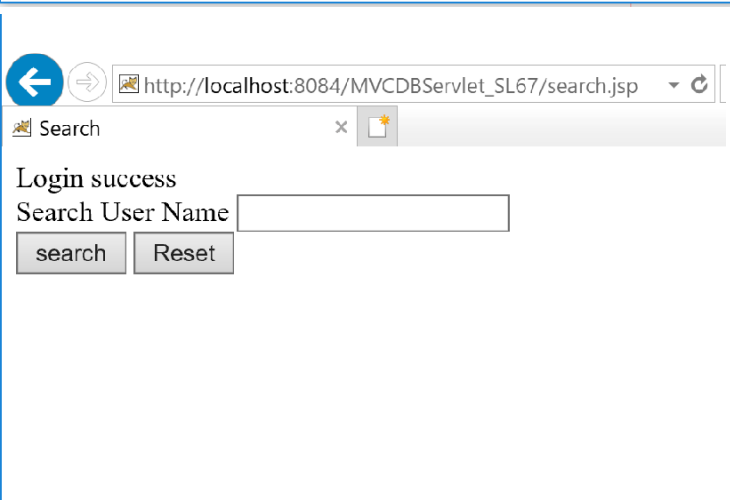
Elements	Description
page Directive	Defines attribute of JSP page and passes processing information about JSP page to JSP container
taglib Directive	Defines custom tags in JSP page
<jsp:param>	Adds value of parameter to a request sent to another JSP page using <jsp:include> or <jsp:forward>
<jsp:forward>	Forwards request from a client to the Web server
<jsp:include>	Includes the output from one file into the other
<tagPrefix:name>	Accesses the functions of custom tags
<jsp:setProperty>	Sets the value of a Java Bean
<jsp:getProperty>	Includes the bean property and value into the result set
<jsp:plugin>	Uses a plugin to execute an applet or Bean
<jsp:useBean>	Sets the location and initializes the bean with a specific name and scope

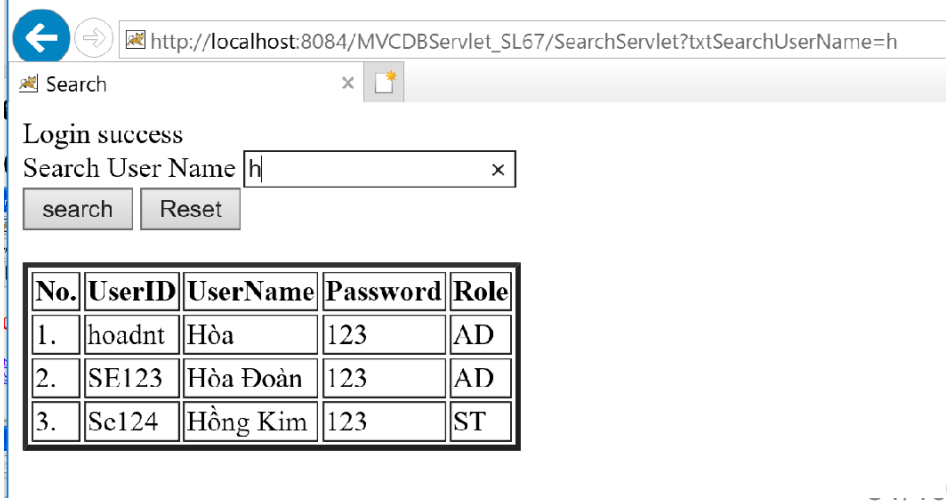
MVC Design Pattern

MVC Model 2









MVC Design Pattern

MVC Model 2

LoginServlet.java

Source History

```
18
19  /**
20   *
21   * @author HD
22   */
23  public class LoginServlet extends HttpServlet {
24      private final String searchPage = "search.jsp";
25      private final String invalidPage = "invalid.html";
26
27      /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
36      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException, SQLException {
38          response.setContentType("text/html;charset=UTF-8");
39          try (PrintWriter out = response.getWriter()) {
40              /* TODO output your page here. You may use following sample code. */
41              String button = request.getParameter("btnAction");
42              String url = invalidPage;
43              if (button.equals("Login")) {
44                  String userID = request.getParameter("txtUserID");
45                  String password = request.getParameter("txtPassword");
46                  RegistrationDAO registrationDAO = new RegistrationDAO();
47                  boolean checkLogin= registrationDAO.checkLogin(userID, password);
48                  if(checkLogin) {
49                      url = searchPage;
50                  }
51              }
52  //              sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer.
53              response.sendRedirect(url);
54          }
55      }
56
57      HttpServlet methods. Click on the + sign on the left to edit the code.
103
104 }
105
```

MVC Design Pattern

MVC Model 2

SearchServlet.java

Source | History

```
20
21  /**
22   *
23   * @author HD
24   */
25  public class SearchServlet extends HttpServlet {
26
27      /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
36      private final String searchpage = "search.jsp";
37      private final String showSearchResult = "search.jsp";
38
39      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
40              throws ServletException, IOException, SQLException, ClassNotFoundException {
41          response.setContentType("text/html;charset=UTF-8");
42          String url = searchpage;
43          String searchValue = request.getParameter("txtSearchUserName");
44          try  {
45              if (!searchValue.isEmpty()) {
46                  RegistrationDAO regDAO = new RegistrationDAO();
47                  List<RegistrationDTO> resultList = regDAO.getListUsers(searchValue);
48                  request.setAttribute("SEARCHRESULT", resultList);
49                  url = showSearchResult;
50              }
51          } finally {
52              try {
53                  RequestDispatcher rd = request.getRequestDispatcher(url);
54                  rd.forward(request, response);
                } catch (Exception ex) {
                    ex.printStackTrace();
57              }
58  //              rd.include(request, response);
59          }
60      }
```

MVC Design Pattern

MVC Model 2

search.jsp

Source | History

```
<%--
    Document   : search
    Created on : Apr 14, 2018, 10:09:57 PM
    Author     : HD
--%>

<%@page import="java.util.List"%>
<%@page import="sample.registration.RegistrationDTO"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Search</title>
    </head>
    <body>
        <div>Login success </div>

        <form action="SearchServlet" >
            Search User Name <input type="text" name="txtSearchUserName"/> <br/>

            <input type="submit" value="search" />
            <input type="reset" value="Reset"/>

        </form>
        <br/>
        <%
            String searchValue = request.getParameter("txtSearchUserName");
            if (searchValue != null) {
                List<RegistrationDTO> listUser = (List<RegistrationDTO>) request.getAttribute("SEARCHRESULT");
                if (listUser != null) {
        %>
        <table border="3">
```

MVC Design Pattern

MVC Model 2

search.jsp

Source | History

```
33        <table border="3">
34            <thead>
35                <tr>
36                    <th>No.</th>
37                    <th>UserID</th>
38                    <th>UserName</th>
39                    <th>Password</th>
40                    <th>Role</th>
41                </tr>
42            </thead>
43            <tbody>
44                <%
45                    int count = 0;
46                    for (RegistrationDTO dto : listUser) {
47                %>
48                <tr>
49                    <td>
50                        <%= ++count%>.
51                    </td>
52                    <td>
53                        <%= dto.getUserID()%>
54                    </td>
55                    <td>
56                        <%= dto.getUserName()%>
57                    </td>
58                    <td>
59                        <%= dto.getPassword()%>
60                    </td>
61                    <td>
62                        <%= dto.getRole()%>
63                    </td>
64                </tr>
65                <% } %>
66            </tbody>
```

MVC Design Pattern

MVC Model 2

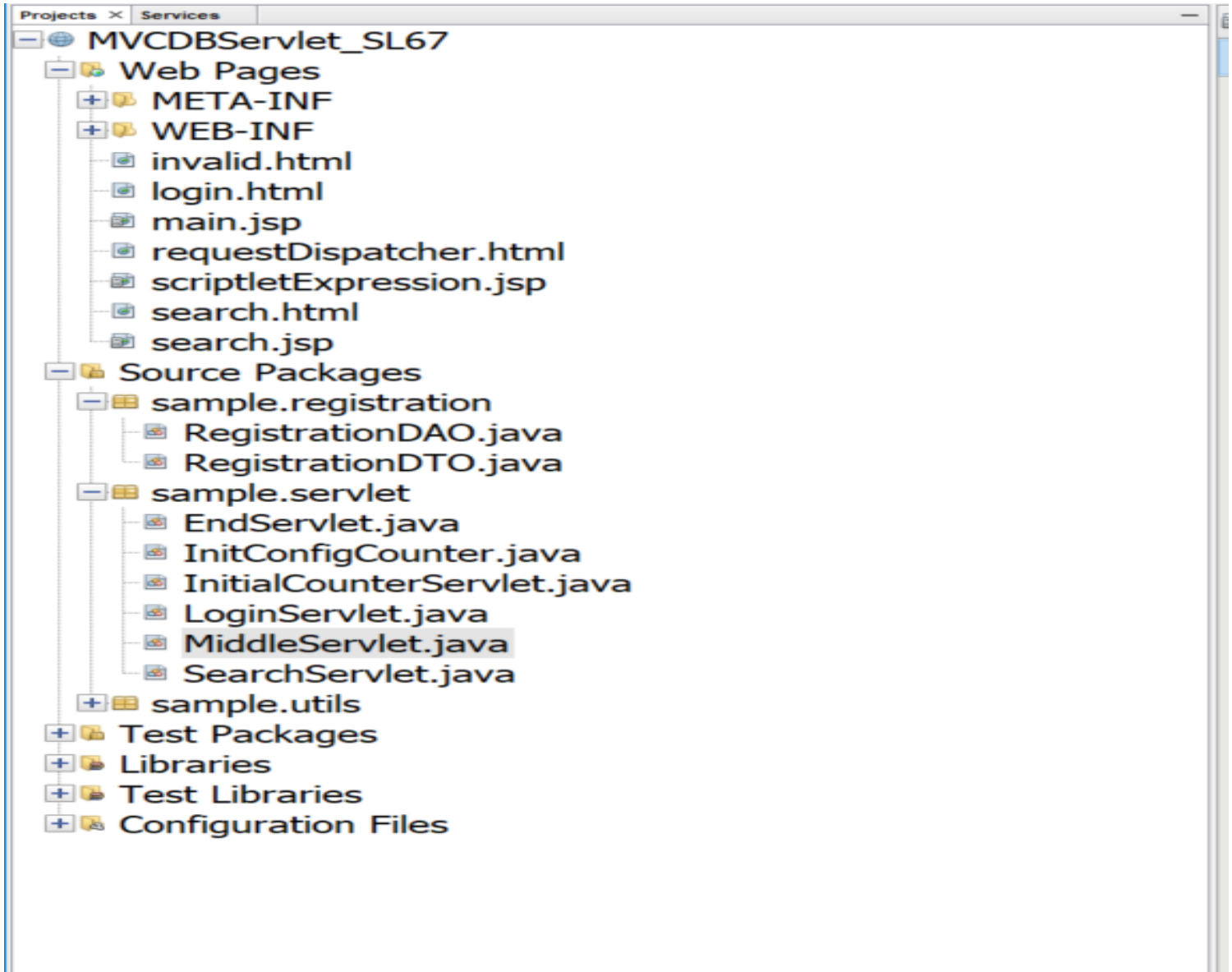
search.jsp

Source | History

```
54                          </td>
55                          <td>
56                              <%= dto.getUserName()%>
57                          </td>
58                          <td>
59                              <%= dto.getPassword()%>
60                          </td>
61                          <td>
62                              <%= dto.getRole()%>
63                          </td>
64                      </tr>
65                      <% } %>
66                  </tbody>
67              </table>
68              <%
69              } else {
70              %>
71              <h2> No record is matched !!</h2>
72              <% }
73                  }
74              %>
75
76
77      </body>
78  </html>
```

MVC Design Pattern

MVC Model 2



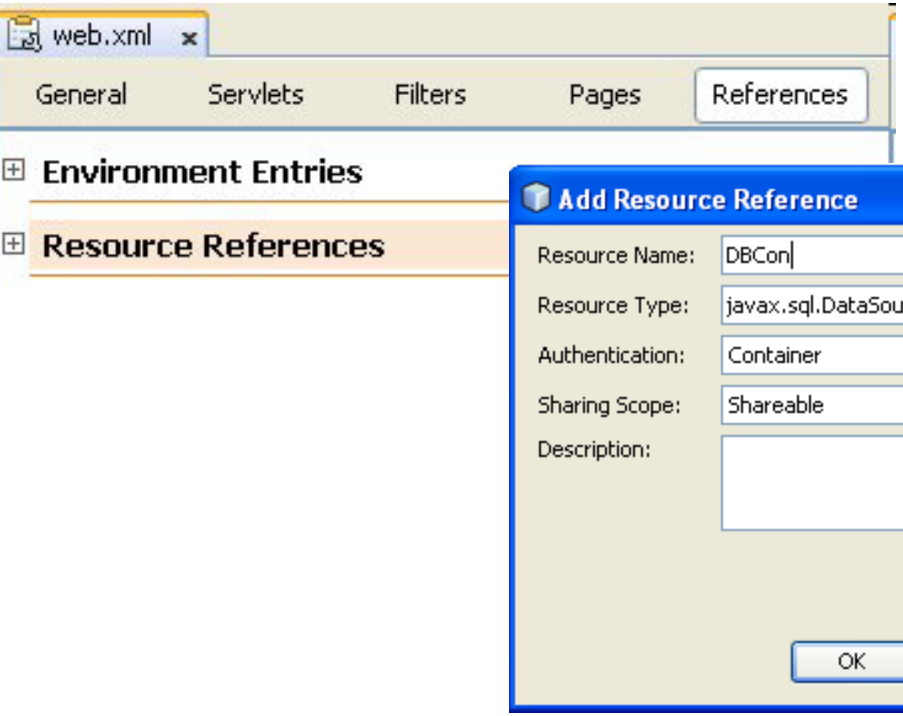
Appendix
Data Source

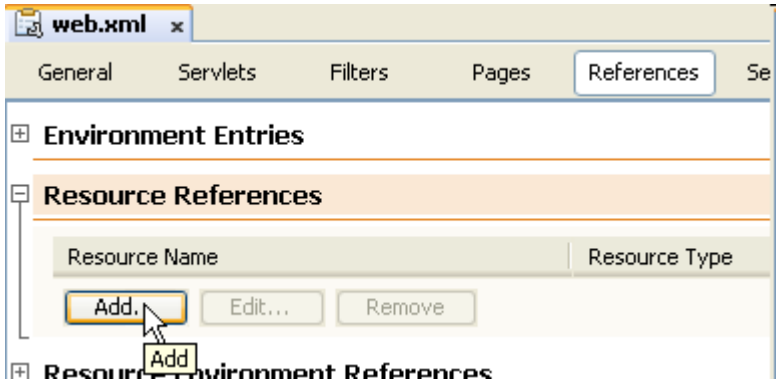
- Java EE applications use **DataSource objects** when they **access relational databases** through the JDBC API
  - A DataSource object **works with** a **JNDI** naming service
  - After it is registered with a JNDI naming service, an application can **use** the JNDI API to **access** that **DataSource object**
- A DataSource has a **set of properties** that **identify** and **describe** the **real-world data source** that it represents
  - **A DataSource XML descriptor** that contains **essential information**, such as the name of the underlying JDBC driver, database URL, the name of the database, and the network protocol, connection pooling properties, and so on
  - A DataSource **alias** is a **logical name mapped** to the name of a **real DataSource available** in the system. It specified in the name element begins with a namespace scope
    - java:comp/env/, the **datasource** will be **available** for the component in which it is **defined**, such as a servlet, EJB, or application client component
  - The **DataSource** alias is used in **application** code to **connect** to the underlying data source

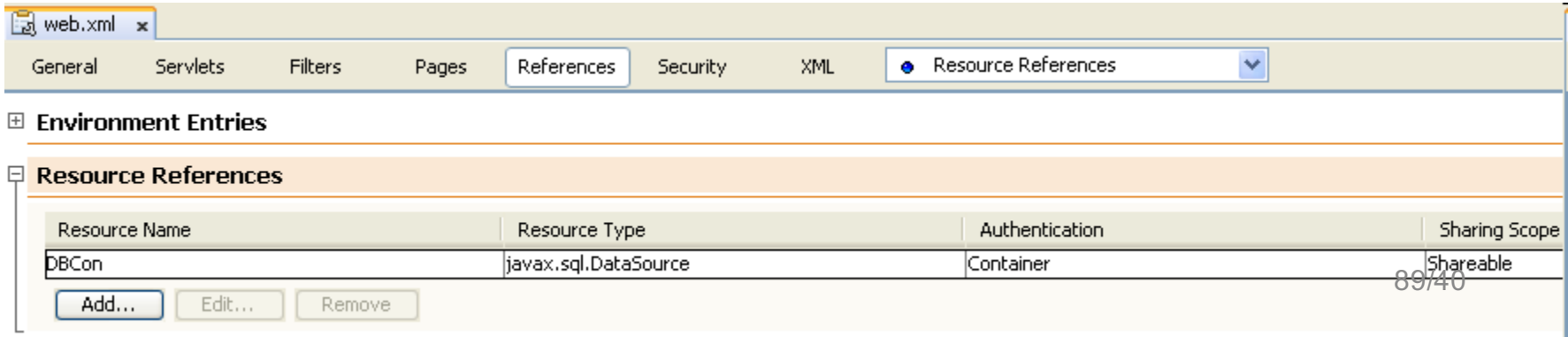
Appendix

Dynamic DB Connection

- **Adding** and **modify** the **web.xml** as following







Appendix

Dynamic DB Connection

- **Adding** and **modify** the **web.xml** as following

web.xml

Source | General | Servlets | Filters | Pages | References | Security

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/
3       <servlet>
7       <servlet-mapping>
11      <session-config>
16      <welcome-file-list>
19      <resource-ref>
20          <res-ref-name>DBCon</res-ref-name>
21          <res-type>javax.sql.DataSource</res-type>
22          <res-auth>Container</res-auth>
23          <res-sharing-scope>Shareable</res-sharing-scope>
24      </resource-ref>
25  </web-app>
```

Appendix

Dynamic DB Connection

- **Adding** and **modify** the **context.xml** in the **META-INF** directory as following

context.xml

Source | History

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <Context antiJARLocking="true" path="/AJDay5_7">
3      <Resource name="KhanhKK" auth="Container" type="javax.sql.DataSource"
4      username="sa" password="trongkhanh"
5      driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriver"
6      url="jdbc:sqlserver://localhost:1433;databaseName=Sinhvien;instanceName=SQL2005"/>
7  </Context>
```

- **Implement code to use**

```
Context ctx = new InitialContext();
Context envCtx = (Context) ctx.lookup("java:comp/env");
DataSource ds = (DataSource) envCtx.lookup("DBCon");
Connection con = ds.getConnection();
```