Advanced
Servlets and JSP

Advanced Servlets Features

- Listeners
- Filters and wrappers
- Request dispatchers
- Security

Listeners

– also called *observers* or *event handlers*

- **`ServletContextListener`**
  - Web application initialized / shut down
- **`ServletRequestListener`**
  - request handler starting / finishing
- **`HttpSessionListener`**
  - session created / invalidated
- **`ServletContextAttributeListener`**
  - context attribute added / removed / replaced
- **`HttpSessionAttributeListener`**
  - session attribute added / removed / replaced

Example: SessionMonitor (1/2)

```
import javax.servlet.*;
import javax.servlet.http.*;

public class SessionMonitor
    implements HttpSessionListener, ServletContextListener {
  private int active = 0, max = 0;

  public void contextInitialized(ServletContextEvent sce) {
    store(sce.getServletContext());
  }

  public void contextDestroyed(ServletContextEvent sce) {}

  public void sessionCreated(HttpSessionEvent se) {
    active++;
    if (active>max)
      max = active;
    store(se.getSession().getServletContext());
  }
```

Example: SessionMonitor (2/2)

```
public void sessionDestroyed(HttpSessionEvent se) {
    active--;
    store(se.getSession().getServletContext());
  }

  private void store(ServletContext c) {
    c.setAttribute("sessions_active", new Integer(active));
    c.setAttribute("sessions_max", new Integer(max));
  }
}
```

Registration in `web.xml`:

```
<listener>
  <listener-class>SessionMonitor</listener-class>
<listener>
```

Filters

- Code being executed before and after the servlet
  - executed in stack-like fashion with servlet at the bottom
- Can **intercept** and **redirect** processing
  - security
  - auditing
- Can **modify requests and responses**
  - data conversion (XSLT, gzip, ...)
  - specialized caching

– all without changing the existing servlet code!

Example: LoggingFilter (1/2)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class LoggingFilter implements Filter {
  ServletContext context;
  int counter;

  public void init(FilterConfig c) throws ServletException {
    context = c.getServletContext();
  }

  public void destroy() {}
```

Example: LoggingFilter (2/2)

```
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
      throws IOException, ServletException {
    String uri = ((HttpServletRequest)request).getRequestURI();
    int n = ++counter;
    context.log("starting processing request #"+n+" ("+uri+")");
    long t1 = System.currentTimeMillis();
    chain.doFilter(request, response);
    long t2 = System.currentTimeMillis();
    context.log("done processing request #"+n+", "+(t2-t1)+" ms");
  }
}
```

Registration of Filters in `web.xml`

```
<web-app ...>
  ...
  <filter>
    <filter-name>My Logging Filter</filter-name>
    <filter-class>LoggingFilter</filter-class>
  </filter>

  <filter-mapping>
    <filter-name>My Logging Filter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  ...
</web-app>
```

Wrappers

- Used by filters to modify requests and responses

- **`HttpServletRequestWrapper`**

- **`HttpServletResponseWrapper`**

- Example: performing *server-side* XSLT transformation for older browsers

Example: XSLTFilter (1/5)

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.jdom.*;
import org.jdom.transform.*;
import org.jdom.input.*;
import org.jdom.output.*;

public class XSLTFilter implements Filter {
  ServletContext context;

  public void init(FilterConfig c) throws ServletException {
    context = c.getServletContext();
  }

  public void destroy() {}
```

Example: XSLTFilter (2/5)

```
public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
    throws IOException, ServletException {
  HttpServletRequest hreq = (HttpServletRequest)request;
  HttpServletResponse hresp = (HttpServletResponse)response;
  boolean client_capable =
    checkXSLTSupport(hreq.getHeader("User-Agent"));
  ServletResponse res;
  if (client_capable)
    res = response;
  else
    res = new BufferingResponseWrapper(hresp);
  chain.doFilter(request, res);
```

Example: XSLTFilter (3/5)

```
    if (!client_capable) {
     try {
       hresp.setContentType("application/xhtml+xml");
       transform(((BufferingResponseWrapper)res).getReader(),
                 response.getWriter());
     } catch (Throwable e) {
       context.log("XSLT transformation error", e);
       hresp.sendError(500, "XSLT transformation error");
     }
   }
 }

 boolean checkXSLTSupport(String user_agent) {
   if (user_agent==null)
     return false;
   return
     user_agent.indexOf("MSIE 5.5")!=-1 ||
     user_agent.indexOf("MSIE 6")!=-1 ||
     user_agent.indexOf("Gecko")!=-1;
 }
```

Example: XSLTFilter (4/5)

```
  void transform(Reader in, Writer out)
      throws JDOMException, IOException {
    System.setProperty("javax.xml.transform.TransformerFactory",
                       "net.sf.saxon.TransformerFactoryImpl");
    SAXBuilder b = new SAXBuilder();
    Document d = b.build(in);
    List pi = d.getContent(new org.jdom.filter.ContentFilter
                           (org.jdom.filter.ContentFilter.PI));
    String xsl = ((ProcessingInstruction)(pi.get(0)))
                 .getPseudoAttributeValue("href");
    XSLTransformer t = new XSLTransformer(xsl);
    Document h = t.transform(d);
    (new XMLOutputter()).output(h, out);
  }
}
```

Example: XSLTFilter (5/5)

```
class BufferingResponseWrapper extends HttpServletResponseWrapper {
  CharArrayWriter buffer;
  PrintWriter writer;

  public BufferingResponseWrapper(HttpServletResponse res) {
    super(res);
    buffer = new CharArrayWriter();
    writer = new PrintWriter(buffer);
  }

  public PrintWriter getWriter() {
    return writer;
  }

  Reader getReader() {
    return new CharArrayReader(buffer.toCharArray());
  }
}
```

Request Dispatchers

- Forwarding requests to other resources
- Often used with JSP...

Security – Roles and Authentication

```
<web-app ...>
  ...
  <security-role>
    <role-name>administrator</role-name>
    <role-name>teacher</role-name>
    <role-name>student</role-name>
  </security-role>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Administration</realm-name>
  </login-config>
  ...
</web-app>
```

Security Constraints

```
...
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Restricted Area</web-resource-name>
    <url-pattern>/restricted/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>administrator</role-name>
    <role-name>teacher</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
...
```

Programmatic Security

Useful request methods:

- `getRemoteUser()`
- `isUserInRole(String role)`
- `isSecure()`
- `getAuthType()`
- `getAttribute("javax.servlet.request.X509Certificate")`

Summary

- Servlets closely follow the **request-response** pattern from HTTP

- Features:
  - Multi-threading
  - Declarative configuration
  - Request parsing, including decoding of form data
  - Shared state
  - Session management
  - Advanced code structuring: listeners, filters, wrappers
  - Client authentication, SSL

Advanced JSP Features

- XML version of JSP
- The expression language
- Tag files
- JSTL
- The Model-View-Controller pattern

JSP Pages Are Not XML

```
<html>
  <head><title>JSP Color</title></head>
  <body bgcolor=<%= request.getParameter("color") %>>
    <h1>Hello World!</h1>
    <%! int hits = 0; %>
    You are visitor number
    <% synchronized(this) { out.println(++hits); } %>
    since the last time the service was restarted.
    <p>
    This page was last updated:
    <%= new java.util.Date().toLocaleString() %>
  </body>
</html>
```

- This page generates HTML, not XHTML
- <%...%> is not well-formed XML

XML Version of JSP

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0"
          xmlns="http://http://www.w3.org/1999/xhtml">
  <jsp:directive.page contentType="text/html"/>
  <jsp:scriptlet>
    response.addDateHeader("Expires", 0);
  </jsp:scriptlet>
  <html>
    <head><title>JSP</title></head>
    <jsp:element name="body">
      <jsp:attribute name="bgcolor">
        <jsp:expression>
          request.getParameter("color")
        </jsp:expression>
      </jsp:attribute>
      <h1>Hello World!</h1>
      <jsp:declaration>
        int hits = 0;
      </jsp:declaration>
      You are visitor number
      <jsp:scriptlet>
        synchronized(this) { out.println(++hits); }
      </jsp:scriptlet>
      since the last time the service was restarted.
      <p/>
      This page was last updated:
      <jsp:expression>
        new java.util.Date().toLocaleString()
      </jsp:expression>
    </jsp:element>
  </html>
</jsp:root>
```

- Uses `<jsp:...>`
- No schema seems to be available
- No validation of the output
- No validation of Java code
- but it's there...

The Expression Language

- We want to **avoid explicit Java code** in JSP templates

- The syntax **`${exp}`** may be used in
  - template text
  - attribute values in markup
- The expression may access
  - variables in the various scopes
  - implicit objects, such as `param`
- The usual operators are available

An Expression Example

```
<html>
  <head><title>Addition</title></head>
  <body bgcolor="${param.color}">
    The sum of ${param.x} and ${param.y} is ${param.x+param.y}
  </body>
</html>
```

Tag Files

Define **abstractions** as **new tags**

wrap.tag:

```
<%@ tag %>
<%@ attribute name="title" required="true" %>
<html>
  <head><title>${title}</title></head>
  <body>
    <jsp:doBody/>
  </body>
</html>
```

```
<%@ taglib prefix="foo" tagdir="/WEB-INF/tags" %>
<foo:wrap title="Addition">
  The sum of ${param.x} and ${param.y} is
  ${param.x+param.y}
</foo:wrap>
```

Content as a Value: A New Image Tag

image.tag:

```
<%@ tag %>
<jsp:doBody var="src"/>
<img src="http://www.brics.dk/ixwt/images/${src}"/>
```

```
<%@ taglib prefix="foo" tagdir="/WEB-INF/tags" %>
<foo:image>widget.jpg</foo:image>
```

Declaring Variables: A Date Context Tag

date.tag:

```
<%@ tag import="java.util.*" %>
<%@ variable name-given="date" %>
<%@ variable name-given="month" %>
<%@ variable name-given="year" %>
<% Calendar cal = new GregorianCalendar();
   int date = cal.get(Calendar.DATE);
   int month = cal.get(Calendar.MONTH)+1;
   int year = cal.get(Calendar.YEAR);
   jspContext.setAttribute("date", String.valueOf(date));
   jspContext.setAttribute("month", String.valueOf(month));
   jspContext.setAttribute("year", String.valueOf(year));
%>
<jsp:doBody/>
```

Using the Date Context

```
<%@ taglib prefix="foo" tagdir="/WEB-INF/tags" %>
<foo:date>
  In the US today is
  ${month}/${date}/${year},
  but in Europe it is
  ${date}/${month}/${year}.
</foo:date>
```

Quick Poll Tags (1/2)

```
<%@ taglib prefix="poll" tagdir="/WEB-INF/tags/poll" %>
<poll:quickpoll title="Quickies" duration="3600">
  <poll:question>
    The question has been set to "${question}".
  </poll:question>
  <poll:ask>
     ${question}?
     <select name="vote">
       <option>yes
       <option>no
     </select>
     <input type="submit" value="vote">
  </poll:ask>
```

Quick Poll Tags (2/2)

```
  <poll:vote>
     You have voted ${vote}.
  </poll:vote>
  <poll:results>
     In favor: ${yes}<br>
     Against: ${no}<br>
     Total: ${total}
  </poll:results>
  <poll:timeout>
     Sorry, the polls have closed.
  </poll:timeout>
</poll:quickpoll>
```

See the tag files in the book...

Tag Libraries

- Libraries of tags capturing common patterns:
  - pagination of large texts
  - date and times
  - database queries
  - regular expressions
  - HTML scraping
  - bar charts
  - cookies
  - e-mail
  - WML
  - ...

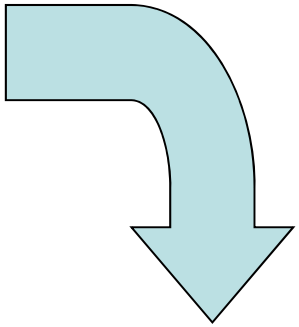
JSTL 1.1

- JSP Standard Tag Library covers:
  - assigning to variables
  - writing to the output stream
  - catching exceptions
  - conditionals
  - iterations
  - URL construction
  - string formatting
  - SQL queries
  - XML manipulation

Selecting Some Recipes

- [x] Beef Parmesan with Garlic Angel Hair Pasta
- [ ] Ricotta Pie
- [x] Linguine Pescadoro
- [x] Zuppa Inglese
- [ ] Cailles en Sarcophages

Select



Title	Calories	Fat	Carbohydrates	Protein	Alcohol
Beef Parmesan with Garlic Angel Hair Pasta	1167	23%	45%	32%	0%
Linguine Pescadoro	532	12%	59%	29%	0%
Zuppa Inglese	612	49%	45%	4%	2%

Using JSTL for the Menu

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<c:import url="http://www.brics.dk/ixwt/recipes.xml" var="xml"/>
<x:parse xml="${xml}" var="recipes" scope="session"/>
<html>
  <head><title>Select Some Recipes</title></head>
  <body>
    <form method="post" action="show.jsp">
      <x:forEach select="$recipes//recipe">
        <c:set var="id"><x:out select="@id"/></c:set>
        <input type="checkbox" name="selected" value="${id}"/>
        <x:out select="title/text()"/>
        <br/>
      </x:forEach>
      <input type="submit" value="Select"/>
    </form>
  </body>
</html>
```

Using JSTL for the Table (1/3)

```
<html>
 <head><title>Nutrition Overview</title></head>
 <body>
  <table border="1">
   <tr>
    <td>Title</td>
    <td>Calories</td>
    <td>Fat</td>
    <td>Carbohydrates</td>
    <td>Protein</td>
    <td>Alcohol</td>
   </tr>
```

Using JSTL for the Table (2/3)

```
<x:forEach select="$recipes//recipe">
  <c:forEach var="id" items="${paramValues.selected}">
   <x:if select="@id=$id">
    <tr>
     <td>
      <x:out select=".//title"/>
     </td>
     <td align="right">
      <x:out select=".//nutrition/@calories"/>
     </td>
     <td align="right">
      <x:out select=".//nutrition/@fat"/>
     </td>
     <td align="right">
      <x:out select=".//nutrition/@carbohydrates"/>
     </td>
```

Using JSTL for the Table (3/3)

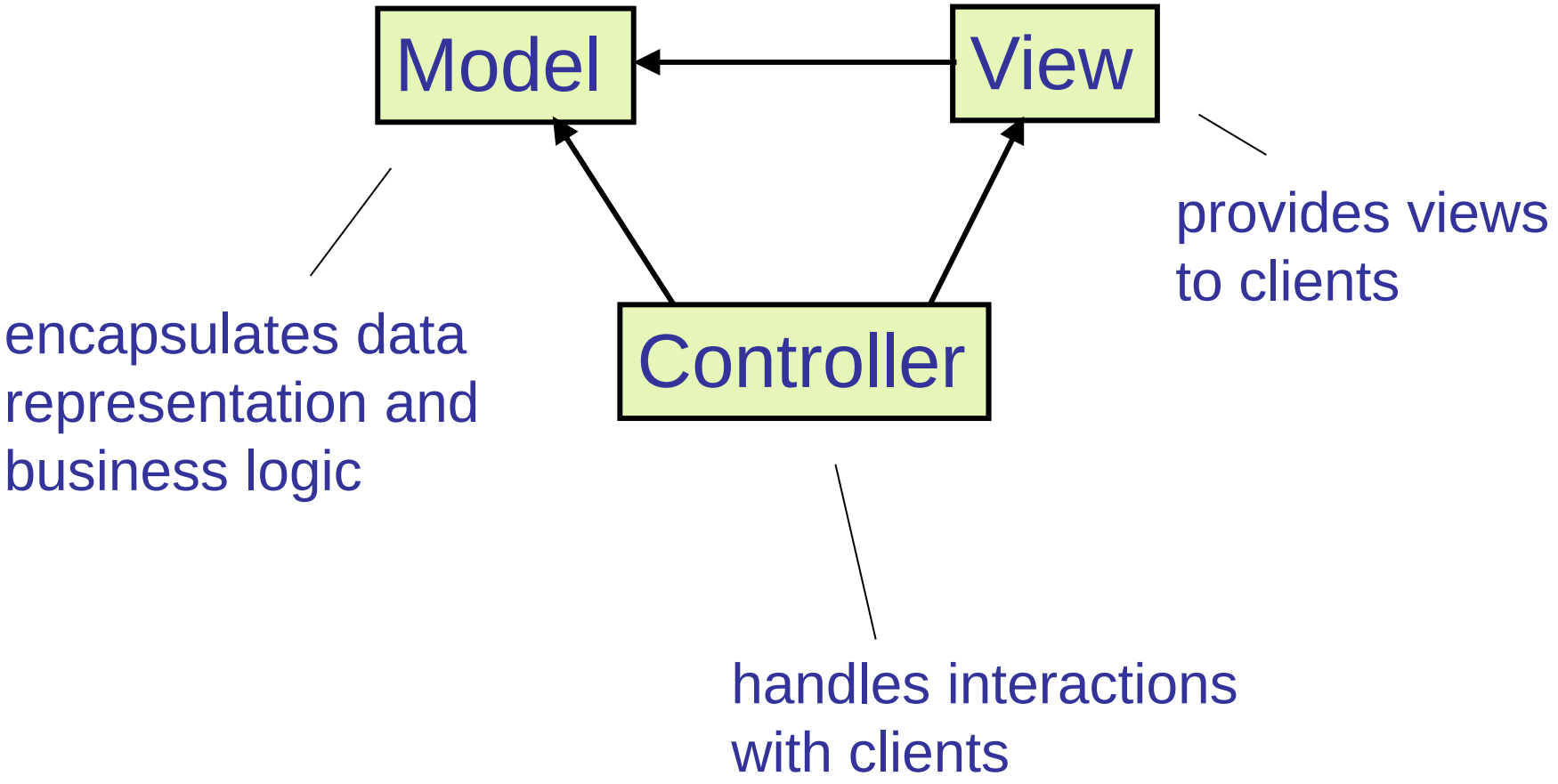
```
      <td align="right">
        <x:out select=".//nutrition/@protein"/>
       </td>
       <td align="right">
        <x:out select=".//nutrition/@alcohol"/>
        <x:if select="not(.//nutrition/@alcohol)">
         0%
        </x:if>
       </td>
      </tr>
     </x:if>
    </c:forEach>
   </x:forEach>
  </table>
 </body>
</html>
```

Evaluation of Tags

- Make Web applications available to a wider range of developers
- May be used to structure applications

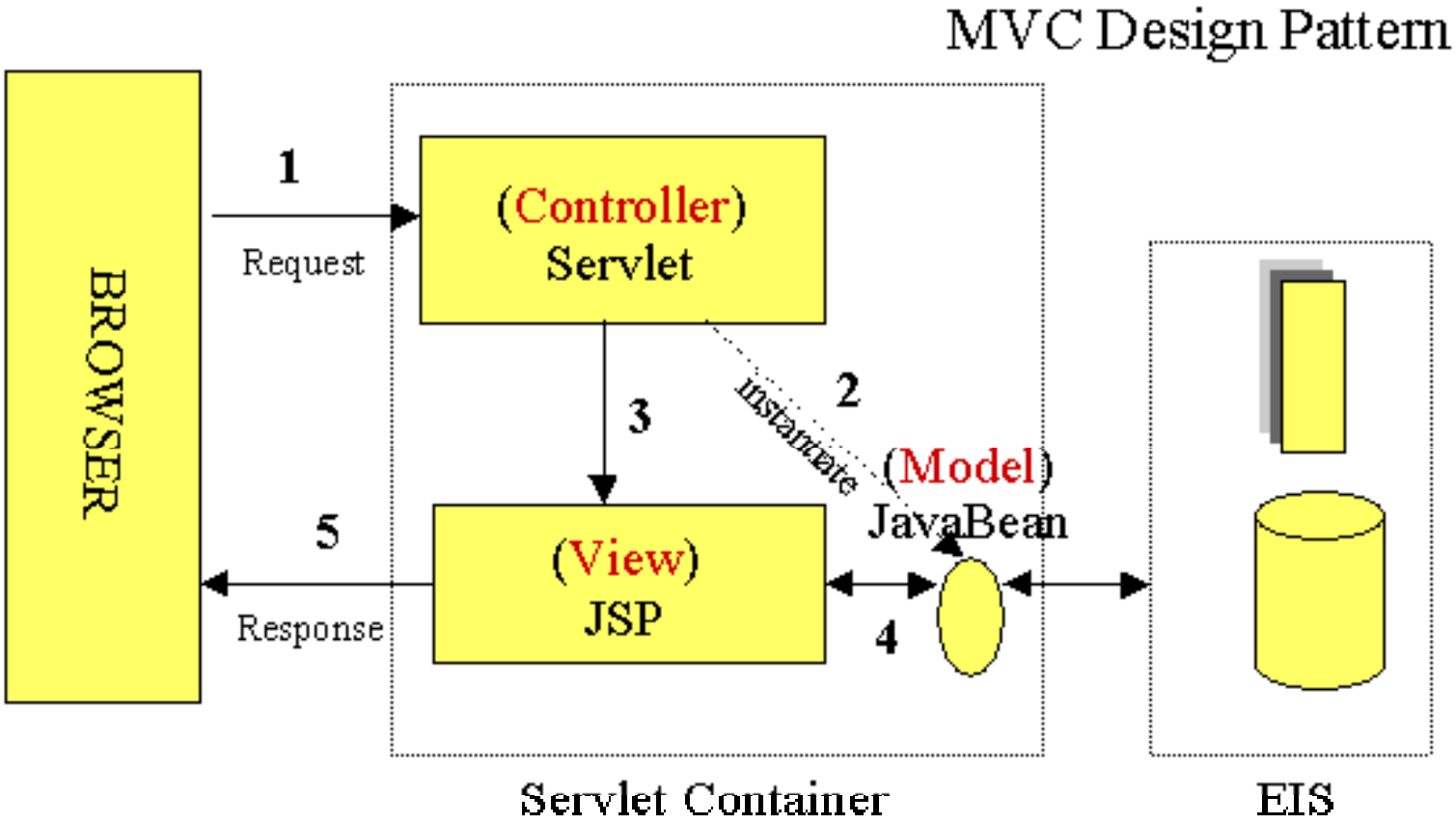
- A myriad of domain-specific languages
- Brittle implementation, hard to debug

The Model-View-Controller Pattern



Model 2 Architecture

- Model 2 is an MVC architecture



The Benefit of MVC

Separation of concerns!

(high cohesion – low coupling)

Using MVC

- Controller: one servlet
- View: JSP pages
- Model: pure Java (e.g. JavaBeans)

[Example in the book: Business Card Server]

Summary

- **JSP templates** are HTML/XHTML pages with embedded code
- The simple **expression language** is often sufficient in place of full-blown Java code
- **Tag files and libraries** allow code to be hidden under a tag-like syntax
- **MVC** provides separation of programming and HTML design tasks