# Web Development using Java, JSP, and Web Services

## Custom Tag Libraries

Lecture #6 2008

1. Web Development
   - Role-Based Development
   - Code Reuse
2. Tag Libraries
   - Custom Tags
   - Tag Library Descriptor
   - Tag Lifecycle

# Web Development

- Web interfaces viewed as user interfaces
- Development methodologies derived from software development
- Development done in roles and teams

# Role-Based Development

- Development role assignments based on competences
- Teams formulated to encompass all roles
- Team leaders divide work into packages and plan projects
- Unique competences often timeshared between teams
- Team size adapted to project size and time / cost restraints

# Typical Roles

- Designer / Systems Architect
- Systems Developer / Programmer
- Interface / Web Developer
- Database Developer
- Graphics Artist
- Usability Specialist
- Team / Project Leader
- Documentation Specialist

# Web Development Scenario

- Project Leader assembles system requirements
- Systems Designer / Architect designs system
- Database Developer designs and implements database
- Systems Developer / Programmer develops database access and logic components
- Interface / Web Developer develops user interfaces (web pages)
- Graphics Artist creates illustration and web site graphics
- Usability Specialist verifies web site usability features
- Documentation Specialist documents all technical solutions

What (if any) of these can be done in parallel?

# Code Reuse

- Use of existing software, or software knowledge, to build new software
- Supports interface abstraction, modularity, loose coupling, information hiding, and separation of concerns
- Code is reused as
  - replicated code (copy and paste)
  - templates
  - generated code
  - functions / procedures
  - modules
  - libraries

# Motivations for Code Reuse

- Expensive to reinvent the wheel
- The wheel not always round in subsequent versions...
- Often (much) faster to built on standardized solutions
- Complex implementations will contain bugs
- Bugs and security vulnerabilities addressed faster by specialized tools manufacturers
- Module interfaces abstract implementation complexity

# Code Reuse in JSP

- Code and script snippets imported (scriptlets)
- Code library use in JSP pages (classes in JARs)
- Reuse of logic layer components (beans and EJBs)
- Tag libraries (classes with tag interfaces)

# Tag Libraries

- Introduced in JSP 1.1
- Allows users to create their own JSP tags
- Commonly referred to as *custom tags*
- Emphasizes role-based development methodology (separates creator of tag from user of tag)
- Integrated in JSP (can alter JSP body response stream)
- Used to encapsulate complex logic and reuse code

# Custom Tags

- A Java class implementing the *Tag* interface (boilerplate implementation classes available)
- Defines tag name, tag attributes & tag body interpretation
- Specify a tag description in a XML-based *descriptor* file
- Included in JSP using the `taglib` directive
- Tags may control JSP processing

# Tag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `TagSupport` / `BodyTagSupport`

```
public interface Tag extends JspTag
{
  Tag getParent ();
  void setParent (Tag t);
  void setPageContext (PageContext pc);
  void release ();
  int doEndTag ();
  int doStartTag ();
}
```

# IterationTag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `TagSupport` / `BodyTagSupport`
- Used to iterate body evaluation (repeats until `doAfterBody()` returns `SKIP_BODY`)

```
public interface IterationTag extends Tag
{
  int doAfterBody ();
}
```

# BodyTag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `BodyTagSupport`
- Used to gain access to body content

```
public interface BodyTag extends IterationTag
{
  int doInitBody ();
  int setBodyContent (BodyContent b);
}
```

# SimpleTag Interface

- Introduced in JSP 2.0
- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `SimpleTagSupport`

```
public interface SimpleTag extends JspTag
{
  JspTag getParent ();
  void setParent (JspTag t);
  void setJspBody (JspFragment body);
  void setJspContext (JspContext pc);
  void doTag ();
}
```

# Tag Library Descriptor

- XML-based configuration file
- Provides a mapping from tag names to tag Java classes
- Contains tag library information and tag descriptions
- Tag descriptions direct how the tag is utilized (by the JSP engine, include tag body etc)
- Required, one per tag library

# Descriptor Content

- `name` - tag alias for use in JSP
  (coupled to tag library namespace)
- `tagclass` - fully qualified implementation class name
- `attribute` - tag attributes (optional)
  (used as tag parameters, values delivered as Strings)
  - name - attribute name
  - required - attribute required to process tag flag
  - rtexprvalue - attribute value from JSP expression flag
- `info` - descriptive information about tag (optional)
- `body-content` - tag body processing directives (optional)
  - EMPTY - no tag body
  - JSP - body contains JSP
  - TAGDEPENDENT - tag processes body itself

# Tag Lifecycle

1. Development
   - coding a tag library & writing a descriptor
   - developing JSP pages that uses the tag library
2. Translation - compile time
   - tags and JSP translated to servlets (tag calls inserted)
3. Evaluation
   - request time
   - tag is loaded and methods are called

# Tag Development

- Can be done in a separate environment (only requires access to the J2EE environment)
- Tag development like any other Java development
- Tags are exported in a JAR file
- Tag library descriptor included in JAR file (in the META-INF directory)

# Tag Translation

Compile time

1. A call to `doStartTag()` is inserted in Servlet
2. Tag body is translated (JSP inserted in Servlet)
3. A call to `doEndTag()` is inserted in Servlet

# Tag Evaluation

Request time

1. `setPageContext()` called, page context provided
2. `setParent()` called, page hierarchy established (used for nested tags)
3. `set`*`Attribute`*`()` is called for attributes
4. `doStartTag()` called, return value directs processing
5. Tag body processed (if so instructed by `doStartTag()`)
6. `doEndTag()` is invoked, return value directs processing
7. `release()` is called to release tag resources (so that tag objects can be reused by thread pools)

# Tag Method Return Values

- `doStartTag()`
  - `EVAL_BODY_INCLUDE` - process tag body
  - `EVAL_BODY_BUFFERED` - tag processes body
  - `SKIP_BODY` - do not process tag body
- `doAfterBody()` (`IterationTag`)
  - `EVAL_BODY_AGAIN` - repeat tag body evaluation
  - `SKIP_BODY` - do not repeat tag body evaluation
- `doEndTag()`
  - `EVAL_PAGE` - process rest of JSP
  - `SKIP_PAGE` - do not process rest of JSP

# Tag Example

```
public class SimpleHelloWorldTag extends TagSupport
{
  //-------------------------------------------------------
  public int doEndTag ()
    throws JspException
  {
    try
    {
      JspWriter out = pageContext.getOut();
      out.print("(Simple) Hello world!");
    }
    catch (IOException e)
    {
      throw new JspException(e.getMessage());
    }

    return Tag.EVAL_PAGE;
  }
}
```

# Tag Library Descriptor Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>lecture06</short-name>
  <uri>taglibs/lecture06</uri>
  <description>
    lecture06 taglib
  </description>

  <tag>
    <name>SimpleHelloWorld</name>
    <tag-class>lecture06.tags.SimpleHelloWorldTag</tag-class>
    <body-content>empty</body-content>
  </tag>

</taglib>
```

# Tag JSP Example

```
<%@ taglib uri="/WEB-INF/lecture06-taglib.tld" prefix="lecture06" %>

<html>
<body>

...

<lecture06:SimpleHelloWorld/>

...

</body>
<html>
```

# web.xml

- Web application configuration file
- Specific to Tomcat
- Maps relative tag descriptor URIs to local filenames
- Optional (specify descriptor path in taglib directive)

```
<taglib>
  <taglib-uri>lecture06-taglib.tld</taglib-uri>
  <taglib-location>/WEB-INF/lecture06-taglib.tld</taglib-location>
</taglib>
```

# Summary

- Tab libraries are used to extend the JSP tag set
- Very powerful way to reuse Java code in JSP
- Custom tags can be used as any JSP tag
- Tag behavior determined by tag developer
- Custom tags well suited to hide large logic segments
- Custom tags are never visible to web clients

# Next Time

- Relational Databases & JDBC