Server Side Internet Programming

Objectives

- The basics of servlets
  - mapping and configuration
  - compilation and execution
- Cookies and sessions
- Request and session attributes
- JSP
  - execution, examples
- JSP Expression Language (EL)
- Databases:
  - SQL (Structured Query Language)
  - JDBC (Java Data Base Connectivity)

Dynamic web content

Content generated from CGI or application servers is different from HTTP

Instead of static content you may generate dynamic content

Even when you don’t respond to forms you can use CGI, servlets, JSP, PHP or any server script language to generate dynamic content

But what are servlets?

What are servlets?

Servlets are programs that run on the server side. They work like CGI programs but are partial Java programs following a special recipe and they are run by a container application program

A servlet reads data from the client, generates an answer – by communicating with files, databases or other programs – and sends back the answer to the client

How do servlets run?

- A servlet runs in a *container* according to a specific SUN standard.
- This means that there is a basic servlet class to use as a template.
- There is a *reference implementation container* called *Tomcat* defining how a servlet is to be run.
- There are many application servers that work as servlet containers.
- Resin, Jetty (is a combined http, java servlet and application server), JOnAS, BEA WebLogic, SIP Servlet Application Server, iPlanet, et.c. see: `http://en.wikipedia.org/wiki/List_of_Servlet_containers` for a more actual (accurate) list
- If you run Tomcat you can test your servlets by submitting a form or by simply starting it without any form parameters.

Servlet structure

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTemplate extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException {
      // Use "request" to read incoming HTTP headers
      // e.g. cookies and query data from HTML forms
      // Use "response" to specify HTTP response status
      // code and headers, e.g. content type and cookies
      PrintWriter out = response.getWriter();
      // Use "out" to send the content to the browser
   }
}
```

Simple text generating servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException {
      // container takes care of headers but
      // we change headers:
      response.setContentType("text/plain");
      // and now the content
      PrintWriter out = response.getWriter();
      out.println("Hello World");
   }
}
```

Compiling a servlet

Assuming that you use Tomcat, be sure that the catalina jar path is in CLASSPATH

In Windows, add to CLASSPATH:

```
%CATALINA_HOME%\common\lib\servlet.jar
```

and compile with javac. If CLASSPATH is not defined, it’s OK to add a classpath value when calling javac:

```
javac -classpath %CATALINA_HOME%\common\lib\servlet.jar myServlet.java
```

Manage Tomcat with ant

If you use ant to install Tomcat all environment variables are set and it works in Solaris, Linux, MacOSX and Windows (all variants, haven’t tested windows 7).

In Solaris you do as follows:

```
~> mkdir test
~> cd test
~/test> module add ant
~/test> cp -r /info/DD1335/localTomcat .
~/test> cd localTomcat
```

If the default port is occupied you need to choose another port. Make the change in conf/server.xml

Manage Tomcat with ant . . .

```
localTomcat> ant install
localTomcat> ant compile
localTomcat> ant tomcat
localTomcat> ant stopTomcat
```

When you have modified a servlet you need to restart tomcat. The fastest way is to use the manager web application.
`http://localhost:8080/manager/reload?path=/labbar`
username: `student`, password: `NADA`

Standard Web Application (webapp) Directories

```
webapps/                                   (servlet contexts)
  ROOT/                                    http://host:port/
    examples/                              http://host:port/examples
      WEB-INF/...                          (not visible by http)
    labbar/                                http://host:port/labbar
      test.html, test.jsp foo/bar.html     http://host:port/labbar/test.html
      WEB-INF/...                          (not visible by http)
        web.xml                            someJSPTagLibrary.tld
        classes/
            HelloWoldServlet.java HelloWorldServlet.class
            somepackage/AClassInThePackage
        lib/
            someLibrary.jar
```

A servlet that generates HTML

```
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;
public class HelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest  request,
                    HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
                     "Transitional//EN\">\n";
    out.println(docType + "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<BODY>\n" + "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
}
```

A utility class generating HTML header

```
public class ServletUtilities {
   public static final String DOCTYPE =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">";
   public static String headWithTitle(String title) {
      return(DOCTYPE +
             "\n" + "<HTML>\n" +
             "<HEAD><TITLE>" +
             title + "</TITLE>\n</HEAD>\n");
   }
}
```

A servlet using our utility class

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimplerHelloWWW extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(ServletUtilities.headWithTitle ("Hello WWW") +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
   }
}
```

Mapping a servlet to a path in web.xml

Normally you can access a servlet like

`http://server:port/webapp/servlet/servletName` e.g.

`http://localhost:8080/labbar/servlet/SimplerHelloWWW`

We can configure further mappings in labbar/WEB-INF/web.xml

We can make the servlet accessible from

`http://localhost:8080/labbar/bla.abc`

or why not `http://localhost:8080/labbar/<anything>.abc` ?

(where `<anything>` may be substituted with just anything.

and `http://localhost:8080/labbar/world/<anything>`

Mapping a servlet to a path in web.xml . . .

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
   <servlet>
      <servlet-name>hello</servlet-name>
      <servlet-class>SimplerHelloWWW</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>hello</servlet-name>
      <url-pattern>*.abc</url-pattern>
   </servlet-mapping>
   <servlet-mapping>
      <servlet-name>hello</servlet-name>
      <url-pattern>/world</url-pattern>
   </servlet-mapping>
</web-app>
```

Calling a servlet from a form

```
<html>
  <body>
    <form action="/servlet/ThreeParams">
      First Parameter: <input type="text" name="param1"><br />
      Second Parameter: <input type="text" name="param2"><br />
      Third Parameter: <input type="text" name="param3"><br />
      <center>
        <input type="submit">
      </center>
    </form>
  </body>
</html>
```

Responding to the form

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    out.println(ServletUtilities.headWithTitle(title) +
       "<body bgcolor=\"#fdf5e6\">\n" +
       "<h1 align=\"center\">" + title +
       "</h1>\n" + "<ul>\n" + " <li><b>param1</b>: " +
       request.getParameter("param1") + "\n" +
       " <li><b>param2</b>: " + request.getParameter("param2") +
       "\n" + " <li><b>param3</b>: " +
       request.getParameter("param3") + "\n" + "</ul>\n" +
       "</body>\n</html>");
   }
}
```

Request details:

`javax.servlet.http.HttpServletRequest`

- Beware, some of the `HttpServletRequest` methods are declared in its superinterface `javax.servlet.ServletRequest`.
  - Look there as well! There are no other subclasses of `javax.servlet.ServletRequest` but you might want to implement one if you use another protocol than HTTP
- As in CGI, you have access to all the HTTP command elements
  - `getMethod()`
  - `getProtocol()`
  - `getQueryString()`

  irrespective of if you used `GET` or `POST` or combined!
- The query string is already broken down into parameters
  - `String getParameter(String name)` returns one parameter
  - `String[] getParameterValues()` returns all values of a parameter
  - `Map getParameterMap()` returns all parameters
- TCP information (like CGI)
  - Local: `getServerPort()`, `getServerName()`
  - Remote: `getRemoteHost()`, `getRemoteAddr()`

Request details . . .

- As in CGI there are special methods for HTTP headers that are often used
  - `getContentType()`
  - `getContentLength()`
- Unlike CGI, you can read *any* HTTP header
  - `getHeader(String name)`
  - `Enumeration getHeaderNames()`
  - `getIntHeader(String name)`
- You can get the POST content by invoking
  - `getInputStream()`
  - `getReader()`

Request printing example

```
public class ShowRequestHeaders extends HttpServlet {
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Servlet Example: Showing Request Headers";
      out.println(ServletUtilities.headWithTitle(title) +
         "<body bgcolor=\"#fdf5e6\">\n" + "<h1 align=\"center\">" +
         title + "</h1>\n" +
         "<b>Request Method: </b>" +
         request.getMethod() + "<br />\n" +
         "<b>Request URI: </b>" +
         request.getRequestURI() + "<br />\n" +
         "<b>Request Protocol: </b>" +
         request.getProtocol() + "<br /><br />\n");
```

Request printing example . . .

```
      out.println("<table border=\"1\" align=\"center\">\n" +
         "<tr bgcolor=\"#ffad00\">\n" +
         "<th>Header Name<th>Header Value");
      Enumeration headerNames = request.getHeaderNames();
      while(headerNames.hasMoreElements()) {
         String headerName = (String)headerNames.nextElement();
         out.println("<tr><td>" + headerName);
         out.println(" <td>" + request.getHeader(headerName));
      }
      out.println("</table>\n</body></html>");
   }
   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
   throws ServletException, IOException {
      doGet(request, response);   /* doPost() do as doGet(). */
   }
}
```

Servlet lifecycle

- init(ServletConfig) is called when the servlet is created. Called once!

```
public void init(ServletConfig conf) throws ServletException {
   // initialize the servlet
}
```

- `init(ServletConfig)` calls `init()` with no parameters
- `destroy()` is called just before the server goes down, or when the web application is reloaded
- `destroyservice(ServletRequest, ServletResponse)` is called for each access
- `doGet()`, `doPost()` and `doXxx`
  `service()` will call one of these, depending on the HTTP method

Servlet configuration parameters

```
public class ShowMessage extends HttpServlet {
  private String message;
  private String defaultMessage = "No message.";
  private int repeats = 1;
  public void init() throws ServletException {
    ServletConfig config = getServletConfig();
    message = config.getInitParameter("message");
    if (message == null) { message = defaultMessage; }
    try {
      String repeatString = config.getInitParameter("repeats");
      repeats = Integer.parseInt(repeatString);
    }
    catch(NumberFormatException nfe) {
    /* NumberFormatException handles case where repeatString is null
      *and* case where it is in an illegal format. */
    }
  }
```

Servlet configuration parameters . . .

```
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
  throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "The ShowMessage Servlet";
    out.println(ServletUtilities.headWithTitle(title) +
                "<body bgcolor=\"#fdf5e6\">\n" +
                "<h1 align=\"center\">" +
                title + "</h1>");
    for(int i = 0; i < repeats; i++) {
      out.println("<b>" + message + "</b><br />");
    }
    out.println("</body></html>");
  }
}
```

Setting configuration parameters in web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.or g/2001/XMLSchema-instance"
   xsi:schemaLocation=
     "http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
   <servlet>
      <servlet-name>Show Msg</servlet-name>
      <servlet-class>ShowMess age</servlet-class>
      <init-param>
         <param-name>message</param-name>
         <param-value>Shibboleth</param-value>
      </init-param>
      <init-param>
         <param-name>repeats</param-name>
         <param-value>5</param-value>
      </init-param>
   </servlet>
</web-app>
```

Cookies

- HTTP is a stateless protocol
  - Even if two HTTP accesses are made by the same user from the same browser, there is nothing in the HTTP accesses to indicate this.
- Cookies can be used to link data to a certain browser that has accessed the site before
  - Name/password (careful! Better to just keep info that the user is already authenticated),
  - the browser has connected to the site before the actual occasion
  - shopping basket, etc
- The cookie is created by the server and sent to the browser with the response
  - Besides application-specific data, each cookie has an expiration date
- The cookie is then kept on the client machine
- It is sent to the server at each subsequent access until it expires
- Cookies have an important application called session tracking
- The rest is not important for the course see
  `http://java.sun.com/javaee/5/docs/api/javax/servlet/http/Cookie.html`
  `http://java.sun.com/javaee/5/docs/api/javax/servlet/http/↩`
  `HttpServletResponse.html#addCookie(javax.servlet.http.Cookie)`

Session tracking

- You often need to keep data about a certain user that's currently accessing the site (the user has a ”session”)
- There are a number of techniques used for session tracking
  - Cookies: may be blocked by the client
  - URL-rewriting: adds a text to the end of every link
  - Hidden forms
- Servlets offer a special mechanism to do session tracking without any worries for the programmer whether they are implemented with cookies or with URL-rewriting
  `javax.servlet.http.Session`
  `request.getSession (boolean create)`
- A session expires if there is no access from the browser for a certain period (typically 15 minutes)
- You may associate attributes with the session.
  The attributes, like the session, will be accessible when responding to all HTTP requests made from the same browser, until the session expires

Session example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;
public class ShowSession extends HttpServlet {
  public void doGet(HttpServletRequest request,
  HttpServletResponse response)
    throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "Session Tracking Example";
      HttpSession session = request.getSession(true);
      String heading;
      Integer accessCount =
        (Integer)session.getAttribute("accessCount");
      if (accessCount == null) {
        accessCount = new Integer(0);
        heading = "Welcome, Newcomer";
      } else {
        heading = "Welcome Back";
        accessCount = new Integer(accessCount.intValue() + 1);
      }
```

Session example . . .

```
      session.setAttribute("accessCount", accessCount);
      out.println(ServletUtilities.headWithTitle(title));
      out.println("<body bgcolor=\"#fdf5e6\">");
      out.println("<h1 align=\"center\">" + heading + "</h1>");
      out.println("<h2>Information on Your Session:</h2>");
      out.println("<table border=\"1\" align=\"center\">");
      out.println("<tr bgcolor=\"#ffad00\">");
      out.println("<th>Info Type</th><th>Value</th></tr>\n");
      out.println("<tr>\n<td>ID</td>");
      out.println("<td>" + session.getId() + "</td>\n</tr>");
      out.println("<tr>\n<td>Creation Time</td>\n<td>");
      out.println(new Date(session.getCreationTime()) + "</td>\n</tr>");
      out.println("<tr>\n<td>Time of Last Access</td>\n<td>");
      out.println(new Date(session.getLastAccessedTime())");
      out.println("</td>\n</tr>");
      out.println("<tr><td>Number of Previous Accesses</td>\n<td>");
      out.println(accessCount + "</td>\n</tr>\n</table>");
      out.println("</body></html>");
    }
  }
```

Request and Session Attributes

- A servlet can access attributes stored within the session or within the request.
  - Request attributes are accessible as long as the request exists
- Attributes are like variables:
  - One servlet can set an attribute in the request or session
  - Another servlet (included or forwarded to) can retrieve the attribute value, change it, et.c.
- If the attribute is in the session, future accesses from *the same browser* can read its value (this is how form login is implemented)
  - javax.servlet.ServletRequest:
    - `setAttribute(String name, Object value)`
    - `getAttribute(String name)`
  - javax.servlet.HttpSession
    - `setAttribute(String name, Object value)`
    - `getAttribute(String name)`
- Most of these methods use an internal map (associating objects with String keys)

Java Server Pages (JSP)

- When we use simple servlets we have to use a lot of `println()`
- In JSP all text is sent to the browser without need for `println()`
- If you want to do other things than printing (call methods, etc)
  - you can open a `<% java scriptlet %>` or
  - you can print a `<%= java expression %>`
- So JSP is a mix of HTML (or whatever you send to the browser) and Java
  - A designer who doesn't know Java can still contribute to the layout!
- In Java scriptlets there are some pre-defined (implicit) objects:
  - `request: javax.servlet.http.HttpServletRequest`
  - `response: javax.servlet.http.HttpServletResponse`
  - `session: javax.servlet.http.HttpSession`
  - `pageContext: javax.servlet.jsp.PageContext`
  - `out: javax.servlet.jsp.JspWriter`
  - `application: javax.servlet.ServletContext` (the webapp)
  - `config: javax.servlet.ServletConfig` (argument to `init()`)
- JSP 2.0 doc
  `http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html`

JSP How-to

- You can put JSPs anywhere in a webapp, except inside WEB-INF
  - So wherever you have content (HTML, images) you can have JSPs as well
- Behind the scene, the servlet container (Tomcat) translate JSPs to `.java` files that define servlets which do `request.getWriter().print()` for every HTML string in the page
  - Since JSP pages are servlets as well, they have all the servlet functionality!
- The `.java` file is compiled behind the scene
  - Yet you will see the compilation errors if there are any
  - To find the generated java with `localTomcat` look in `work/Standalone/localhost/labbar`
- With JSP, you do not need to restart Tomcat or reload the servlet context when you change something. It will be re-translated to Java and recompiled for you automatically

JSP vs other technologies

- JSP vs ASP
  - Java is more powerful than Basic and more portable.
  - Lately ASP works with C# but that's still less portable than Java
- JSP vs PHP
  - Java is more widespread and has more industry support
- JSP vs servlets
  - JSP is a servlet in the end, but it can be easier to get things together if there is a lot of HTML layout to generate
  - For more programming-intensive tasks, servlets can work better
- Java Server Faces (JSF) is a more GUI-style web programming and is hard to combine with JSP

The @page directive

```
<%@page import="java.util.*,java.io.*" contentType="text/html"%>
```

- Packages imported by default: `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`
- The default `contentType` is `text/html`
- errorPage: the page to which a forward should occur in case of error
  - The `exception` object will be predefined in that page
  - `isErrorPage` indicates that this page is for dealing with such errors in JSP pages
- `buffer`, `autoFlush` for response content buffering
  - If you know your page will produce a lot of content, it may be important to work on these

Class declarations

- JSPs generate a Java class, a HttpServlet subclass
- Most of the JSP content will go into the service() method
- If you want to have something outside that method (variable declarations or method definitions) you can use `<%! ...   %>`

  -
    ```
    <%! private int accessCount = 0; %>
    Accesses to page since server reboot:
    <%= ++accessCount %>
    ```
  - This will be translated into something like:

    ```
    public class myJsp extends HttpServlet{
      private int accessCount = 0;
      public void service(HttpSerletRequest req,
                          HttpServletResponse resp)
      throws IOException, ServletException{
        ServletOutputStream out= req.getOutputStream();
        out.print("Accesses to page since server reboot:");
        out.print(++accessCount);
      }
    }
    ```

Including and forwarding

- You may include a page or servlet by specifying a path
  - `<jsp:include page="path" />`
  - `path` can be relative (even using `../`, et.c.)
  - or absolute, starting with `/` which represents the root of the webapp
  - you can't include pages from other webapps in the same server
  - `page="/servlet/somePackage.someClass"` will include a servlet
- You can terminate a page and delegate the execution to another page or servlet
  - `<jsp:forward page="path" />`
  - You can also do this in java using the `pageContext` object see the `include()` and `forward()` methods
- You can also include or forward from servlets
  - `request.getRequestDispatcher(String)` will give you a `javax.servlet.RequestDispatcher` that can perform either include or forward
- You can textually include a file in JSP
  - `<%@include file="path" %>` simply copies JSP code from the indicated file into the caller file prior to any code generation
  - While `jsp:include` is analogous to a method call, `@include` corresponds to `#include` in C

Page context attributes

- Page attributes are visible only within the page
  - Not in included or forwarded pages while
  - request attributes are visible in all included and forwarded pages
- `javax.servlet.jsp.PageContext` (the `pageContext` JSP variable) offers a mechanism to manipulate variables from all scopes
  - '`scope`' can be

`PageContext.PAGE_SCOPE`	(page attributes, default)
`PageContext.REQUEST_SCOPE`	(request attributes)
`PageContext.SESSION_SCOPE`	(session attributess)
`PageContext.APPLICATION_SCOPE`	(accessible to all servlets and JSPs within the same webapp)

  - with methods:

    `setAttribute(String name, Object value, int scope)`
    `getAttribute(String name, int scope)`
    `findAttribute(String name)` searches from page to app scope
    `removeAttribute(String name, int scope)`

- Request, session or application attributes can be used to communicate between the page and the pages/servlets it includes or those that included it

Expression Language

allows for simpler page code

- You may write
  `Hi. The surname you sent is ${param.surname}` instead of
  `Hi. The surname you sent is <%= request.getParameter("surname") %>`
- Manipulate attributes and parameters easier
  - `${attrName}` is equal to `pageContext.getAttribute("attrName")`
  - `${param.paramName}` is the same as `${param["paramName"]`
  - `${sessionScope.attrName} sessionScope` and `param` are actually `java.util.Map` objects. In general:
    - `${aMap.aKey}` ≡ `${aMap["aKey"]}` ≡ `${anArray[anIndex]`
    - `${anObject.anItem}` will call `anObject.getAnItem()`
- Expressions with attributes and parameters (`${attr1 + 1 + attr2}`)
  - Arithmetic operators: `+, -, *, /, %, mod, div`
  - Logical operators : `and, or, not, &&, ||, !`
  - Comparison operators :
    `==, eq, <, lt, <=, le, >, gt, >=, ge, !=, ne`
  - `empty`: operator that tests for `""` or null
  - choise: `expr?exprTrue:exprFalse`
    `You sent ${empty(param.message)?"no":"a"} message`

Implicit objects in EL

- `pageContext`: gives access to `session`, `request`, `response`
  - `${pageContext.request.contextPath}`
    Results into a call to `pageContext.getRequest().getContextPath()`
- Attributes: `pageScope`, `requestScope`, `sessionScope`, `applicationScope`
  - `${attribute}` will search all scopes, starting with page
- CGI query string parameters: `param`
  - `paramValues` for multiple-value parameters: `paramValues['a'][0]`
- HTTP headers: `header`
  - `${header.host}` or `${header["host"]}`
  - `headerValues` for multiple-value headers:
    `headerValues['Accept-language'][0]`
- HTTP cookies: `cookie`

Custom JSP tag libraries. JSTL

The @taglib directive

```
<%@ taglib uri="http://www.sometaglibprovider.com/name"
           prefix="special" %>
<html><body>
...
<special:makeBold>this text will become bold
</special:makeBold>
<special:timeOfDay format="hh:MM:ss" />
</body>
</html>
```

Such libraries are implemented using `javax.servlet.jsp.tagext.*`
The most important tag library is the Java Standard Tag Library (JSTL)
The normal JSP page does not normally need the whole power of Java, no need to create classes

Custom JSP tag libraries. JSTL . . .

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt"
           prefix="c" %>
<c:set var="count" value="34" />
 ${count} <%-- Will print 34 --%>
<c:forEach items="${orders}" var="order">
   <c:out value="${order.id}"/>
</c:forEach>
```

`c:forEach` can iterate in any `java.util.Collection`, `Iterator`, `Enumeration`, etc (see Lecture 2)

It can even iterate in arrays!

```
<c:if expr="${empty(param.x)}">
  The x parameter was not specified</c:if>
```

The JSTL/EL version of header printing

```
<%@page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>
<html>
 <head><title>Welcome!</title></head>
<body>
  You are connecting from ${pageContext.request.remoteHost}<br />
  <table border="1">
  <c:forEach items="${pageContext.request.headerNames}"
             var="headerName">
    <tr><td>${headerName}</td><td>${header[headerName]}</td></tr>
  </c:forEach>
  </table>
  <dl><%-- bonus --%>
  <c:forEach items="${paramValues}" var="p">
    <dt>${p.key}</dt>
    <dd><c:forEach items="${p.value}" var="q">${q}
        </c:forEach></dd>
  </dl>
</body>
</html>
```

Comments on JSTL/EL version of header printing

- `paramValues` is a `java.util.Map` so each of its items (`p`) will be of type `java.util.Map.Entry` which contain the methods `getKey()` and `getValue()`
- `${p.key}` results into a call to `p.getKey()`
- `${p.value}` results into a call to `p.getValue()` which in this case returns a `String[]` so we can `<c:forEach >` through it

Three-tier systems

- Three tiers, each running on one machine (or on the same)
- Presentation layer, e.g. Browser (displays user interface from HTML)
- Application Layer, e.g. HTTP server with servlets and JSP
- Data storage layer, e.g. a database management system (SQL)
- Today, most advanced CGI systems (including servlet/JSP) work with a database where they store the data that they display and change

Databases: SQL

- SQL (Structured Query Language) Tutorials
  - texttthttp://www.w3schools.com/sql/default.asp
  - texttthttp://perl.about.com/od/howtousesql/
- Every SQL server has a SQL client where you can type commands
  - `module add psql` on Solaris
  - `psql -h nestor.nada.kth.se (password: <your )`
  - you end up in "your own" database
  - End all commands with ';'
- Creating a table

  -
    ```
    CREATE TABLE course (id integer primary key,
       name varchar(255), credits real)
    ```
  - `id` can be used to uniquely identify the course (primary key)
  - `\d course` shows the structure of the table
  - `\d` shows the names of all tables in the database

Databases: SQL . . .

- Inserting records
  - `INSERT INTO course VALUES (5, 'DD1335', 7.5)`
- Changing fields
  - `UPDATE course set credits=4 WHERE id=5`
  - `UPDATE course set credits=5 WHERE name like 'DD133%'`
- Removing records
  - `DELETE FROM course WHERE id=5`
  - `DELETE FROM course WHERE name='DD1335'`

Simple SQL queries

Q How many credits does course DD1335 have?

A `SELECT credits FROM course WHERE name='DD1335'`

Q Which courses have names starting with 'DD1', and what credits do they have?

A `SELECT name, credits FROM course WHERE name like 'DD%'`

- Make a table with students

  -
    ```
    CREATE TABLE student(id integer, name varchar(255),
    startyear integer)
    ```
- A table with course attendance: a many-to-many relationship

  -
    ```
    CREATE TABLE course_attendance (
      student_id integer, course_id integer,
      lab1 integer, lab2 integer, project integer,
      PRIMARY KEY (student_id, course_id)
    )
    ```

Simple SQL queries . . .

Q Which students joined the course DD1335?

A
```
SELECT s.name
FROM course c, course_attendance ca, student s
WHERE ca.student_id=s.id
AND c.id=ca.course_id and c.name='DD1335'
```

Q How many students joined the course DD1335?

A
```
SELECT count(ca.student_id)
FROM course c, course_attendance ca
WHERE c.id=ca.course_id and c.name='DD1335'
```

Java as SQL client: JDBC

```
import java.sql.*;
public class JDBCCode{
   public static void main(String args[]){
      try {
        Class.forName("org.postgresql.Driver" ); // load driver
      } catch (ClassNotFoundException cfn){
        System.out.println("Problem loading driver");
        System.exit(1);
      }
      try{//Establish a connection to the database, second
         //argument is the name of the user and the third
         //argument is a password (blank)
         Connection con = DriverManager.getConnection(
           "jdbc:postgresql://nestor.nada.kth.se:5432/dbname",
           "username", "userPassword");
         //Create a statement object
         Statement selectStatement = con.createStatement();
```

Java as SQL client: JDBC . . .

```
      //Execute the SQL select statement
      ResultSet rs =
        selectStatement.executeQuery (
          "SELECT name, startyear FROM " +
          "student WHERE name like ’John%’");
      while(rs.next())
        System.out.println ("Name = "+ rs.getString(1) +
                            "Salary = "+ rs.getInt(2));
      selectStatement.close();
      con.close();
      rs.close();
    } catch(Exception e) {
      System.out.println("Problems with access to database");
      e.printStackTrace();
      System.exit(2);
    }
  }
}
```

JDBC: changing data

```
try {
  Class.forName(driverName);
}
catch (ClassNotFoundException cfn) {
  //Problem with driver, error message and return
  //to operating system with status value 1
  System.out.println("Problem loading driver");
  System.exit(1);
}
try {
  Connection con = DriverManager.getConnection(
           "jdbc:postgresql://nestor.nada.kth.se:5432/dbname",
           "username", "userPassword");
  Statement updateStatement = con.createStatement();
  String studentName ="...", startYear = "...";
```

JDBC: changing data . . .

```
  int noOfRows =
    updateStatement.executeUpdate (
      "INSERT INTO student (name, startYear)" +
      "VALUES(" + "’" + studentName + "’," + startYear);
     //Execute the SQL insert statement
} catch(SQLException sqe) {
      System.out.println("Problem updating database");
}
```

JDBC . . .

Sun's JDBC tutorial

`http://java.sun.com/docs/books/tutorial/jdbc/index.html`

More info on the net

`http://www.javaworld.com/javaworld/jw-051996/jw-05-shah.html`
`http://www.javaworld.com/channel_content/jwjdbc-index.shtml`