Network Computing Lecture 14

Professor Louise E. Moser

Spring 2012

More on Java Servlets

- Java Servlets are stateful objects on the server-side that support the request / response computing model
  - The request and the response reflect the states of the client and the server at the time of the client's request
- Servlets can be plugged into a Java-enabled Web Server to provide custom services, such as:
  - Dynamic (runtime) changes to content and / or presentation of HTML Web pages that are tailored for each visitor or each hit
  - New features and new (standard or custom) protocols
- Servlets can
  - Act as a proxy for a client
  - Augment the features of the middle tier
  - Perform application processing in the middle tier

GET and POST Requests

- As we have already seen, the main functionality of a Servlet is coded in the ***service()*** method, or in one of several special service methods, which include
  - ***doGet()*** - Replies to a request from the client by sending back an HTML page and the form that the client is to use
    - The Servlet *doGet()* method maps directly to the HTTP *GET* request
  - ***doPost()*** - Receives the form data that the client submits, does some processing, and sends back an HTML page to the client
    - The Servlet *doPost()* method maps directly to the HTTP *POST* request

GET Request

- The HTTP GET request asks for information from a Web Server
- The information could be a file, output from a device, or output from a program, such as a Servlet
- On most Web Servers, Servlets are accessed via a URL that starts with **/servlet/**
- For example:

  *GET /**servlet**/MyServlet**?**user=john+dough**&**company=IBM%20INC HTTP/1.1*
  Connection: Keep-Alive
  User-Agent: Mozilla/4.0 (compatible; MSIE 4.0l; Windows NT)
  Host: www.google.com
  Accept: image/gif, image/x-xbitmap, impage/jpeg, image/pjpeg

- The URL in the above *GET* has a list of name / value (parameter / value) pairs following the **?**
- The name / value pairs are separated by an **&**

URL Encoding

- The last character of a URL is the first blank space character encountered after the beginning of the URL
- To ensure that all name / value pairs are sent to the Web Server, there must be no blank space characters in the data
  - Thus, the browser substitutes the + character for a blank space in the data
  - In the previous example, the blank space between *john* and *dough* is replaced by +
- The browser also replaces special characters (like punctuation marks) with the **%** character, followed by the hexadecimal value of those characters in the ASCII code set
  - In the previous example, a blank space is replaced by %20
- Such substitutions are referred to as **URL encoding**

POST Request

- The HTTP *POST* request allows a client to send data to the server
- *POST* can be used for several purposes:
  - Passing more information than GET allows (which is typically limited to a few hundred bytes)
  - Posting information to a news group
  - Adding entries to a Web site's name-address book
- For example:

*POST /**servlet**/MyServlet HTTP/1.1*
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0l; Windows NT)
Host: www.google.com
Accept: image/gif, image/x-xbitmap, impage/jpeg, image/pjpeg
Content-type: application/x-www-form-urlencoded
Content-length: 33

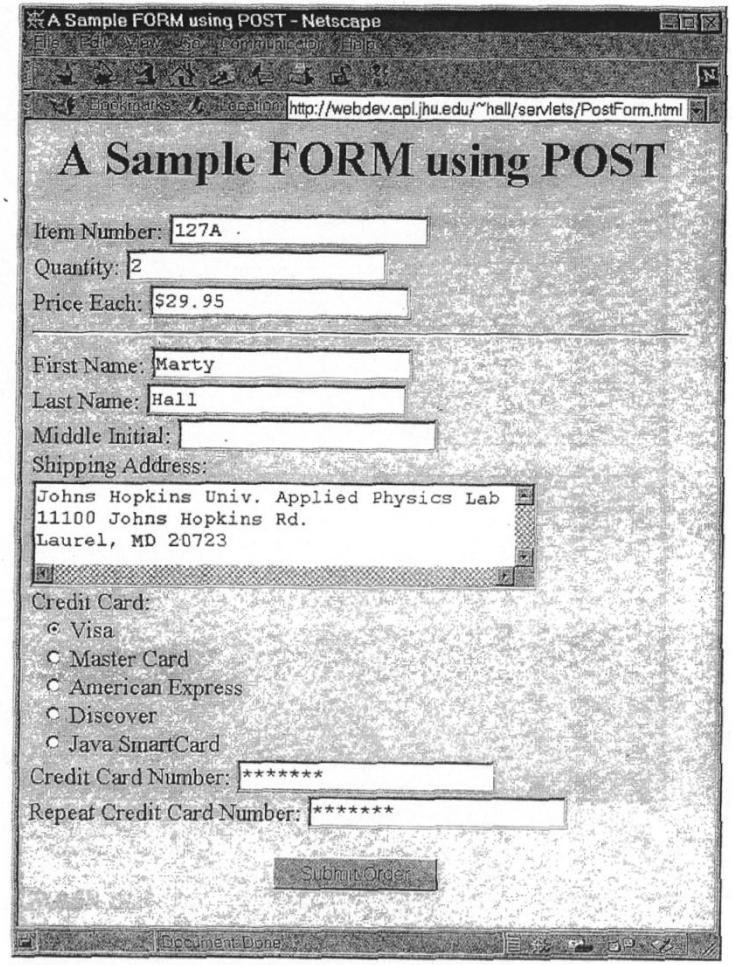
user=John+Dough&company=IBM%20INC

POST Request (cont)

- Note the blank line, which signals the end of the *POST* request header and the beginning of the name / value pairs
- *POST* does NOT support sending parameters encoded within a URL, as *GET* does
- Because multiple clients can try to update data simultaneously, you must use **synchronized** methods when you use *doPost()* in Java

A Sample HTML Form Using POST

- See the HTML for the form in the handout, i.e.,
  <FORM ACTION= "/servlet/hall.ShowParameters" METHOD="POST">



Reading Form Data

- As we have seen, a service **request** can contain information in the form of name / value pairs, as a *ServletInputStream* or a *BufferedReader*
- To get that information, use
  - *getParameter()* - Returns a value corresponding to a particular parameter name
  - *getParameterNames()* - Returns an array of strings with the names of all of the parameters
  - *getParameterValues()* - Returns an array of strings with the values of all of the parameters
- As we have also seen, a service **response** uses a *ServletOutputStream* or a *PrinterWriter* to reply to a client

Example: Three Parameters Servlet

- The three parameters in this example are defined in the following form and set by the client in the GUI

```
<FORM ACTION="/servlet/cwp.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>
```

Reading Three Request Parameters

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

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
      throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String title = "Reading Three Request Parameters";
```

Reading Three Request Parameters (cont)

```
out.println(ServletUtilities.headWithTitle(title)
            + "<BODY>\n"
            + "<H1  ALIGN=CENTER>" + title + "H1>\n"
            + "<UL>\n"
            + " <L1>param1: "
            + req.getParameter("param1") + "\n"
            + " <L2>param2: "
            + req.getParameter("param2") + "\n"
            + " <L3>param3: "
            + req.getParameter("param3") + "\n"
            + </UL>\n"
            </BODY></HTML>");

 public void doPost(HttpServletRequest req,
                    HttpServletResponse res)
        throws ServletException, IOException {
   doGet(req, res);
}
```

Reading Three Request Parameters (cont)

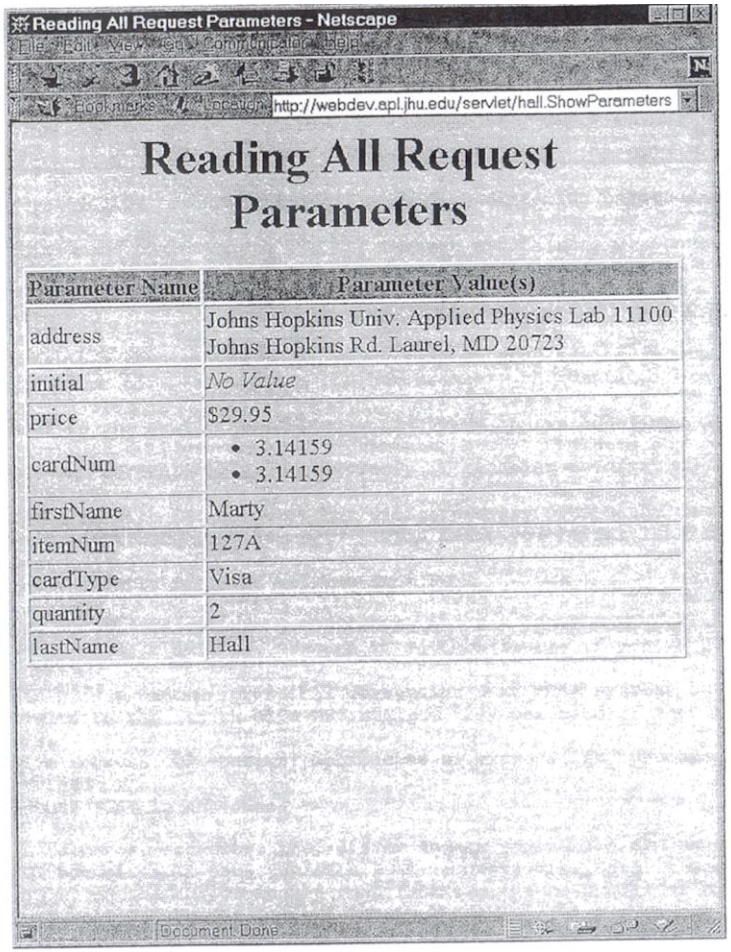
- Note that *doGet()* is nested inside *doPost()*
- Doing this is considered good practice because it permits flexibility for the client and requires little extra programming

Show Parameters Servlet

- See the code for the *ShowParameters* Servlet in the handout
- Note the use of the packages

  import javax.servlet.;*

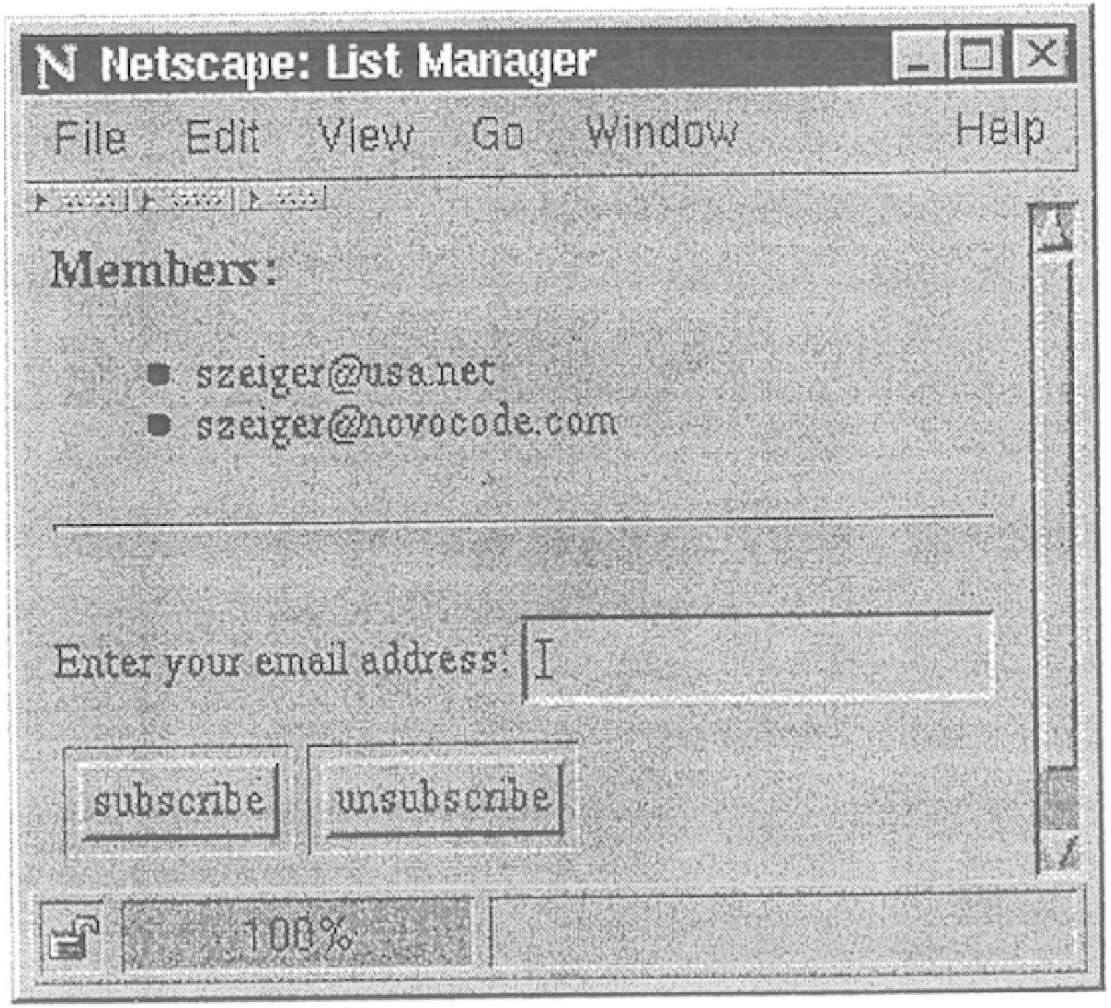
  *import javax.servlet.http.**



Example: List Manager Servlet

- See the code for the *ListManager* Servlet in the handout
- The *doGet()* method replies to *GET* requests by sending an HTML page that contains
  - The list of the currently subscribed member addresses, and
  - The form that the client must use to subscribe or unsubscribe an address
- Note that the response is marked as not cacheable to proxy servers and clients, because it is dynamically created
- Note that the form asks the client to use *POST* for submitting form data

List Manager Servlet (cont)



List Manager Servlet (cont)

- The *doPost()* method receives the submitted form data, updates the address list, and sends back a confirmation page
- First, it retrieves the form parameters "*email*" and "*action*" using the *getParameter()* method
- Then, it calls *subscribe()* or *unsubscribe()*
- When a user error occurs, it uses *res.sendError()* to send back an error response with a *BAD REQUEST* response code
- It uses *req.getRequestURI()* to get the URI of the Servlet for a link back to the main page, which is created by *doGet()*
- Finally, it sends back a confirmation page

Server Side Includes

- To support Web pages that have a small amount of dynamic information, some Web Servers support a concept called **Server Side Includes (SSI)**
- To use SSI, you must use a special file extension (*.shtml*), which tells the Web Server that it should look for SSI tags in the file
- Sun's Java Web Server defines a special SSI tag called the ***<servlet>*** tag
- For example,

```
<servlet code="DatePrintServlet">
    <param name=timezone  value=pst>
</servlet>
```

- This tag causes a Servlet named *DatePrintServlet* to be invoked to generate some inline content
- For SSI, the output MIME type is set to *text/plain* (rather than *text/html*), because only part of the HTML page is generated

Sessions

- A **session** is a continuous connection from a particular client browser to a server during some finite time interval
- The time interval is usually configurable from the Web Server (for Sun's Java Web Server, the default is 30 minutes)
- HTTP Servlets allow you to maintain session information using the *HttpSession* class
- The *HttpServletRequest* object allows you to get a session using the *getSession()* method, which has a boolean parameter
  - If the boolean parameter is true, a new session is created
  - If not, a null is returned

Sessions (cont)

- Once you have access to an *HttpSession* object, you can invoke the following methods on it
  - *getCreationTime()* - Returns the creation time of the session
  - *getLastAccessedTime()* - Returns the time the last request for the session was sent
  - *putValue(key, value)* - Stores session-specific information along with the given key
  - *getValue(key)* - Retrieves session-specific information corresponding to the particular key

Using Servlets with JDBC

- Servlets often connect to a database using JDBC
- Servlets provide better access control to the database, because they allow only the middle tier to communicate with the database
- You can set up multiple database connections in a pool when a Servlet is initialized, and share the database connections in the pool among many client requests

Using Servlets with JDBC (cont)

- In the ***init()*** method, connect to the database

```
Connection conn = null;

public void init(ServletConfig cfg)
        throws ServletException {

  super.init(cfg);

  // Load driver
  String name = cfg.getInitParameter("driver");
  Class.forName(name);

  // Get connection
  conn = DriverManager.getConnection(urlString);
}
```

Using Servlets with JDBC (cont)

- In the ***doGet()*** method, retrieve database information

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
        throws ServletException, IOException {
  res.setContentType("text/html");
  // Have browser ignore cache - force reload
  res.setheader("Expires",
                "Sun, 01 June 2008 00:00:090 GMT");
  Statement stmt = null;
  ResultSet result = null;
```

Using Servlets with JDBC (cont)

```
try {
    // Submit query
    stmt = conn.createStatement();
    result = stmt.executeQuery(
                    "SELECT programmer, cups "
                    + "FROM JoltData ORDER BY cups DESC;");

    // Create output
    PrintWriter out = res.getWriter();
    while (result.next()) {
        // Insert code for generating output from the ResultSet
    }
  }
  finally {
    if (result != null) result.close();
    if (stmt != null) stmt.close();
  }
  out.flush();
  out.close();
}
```

Using Servlets with JDBC (cont)

- In the ***destroy()*** method, close the connection to the database

```
public void destroy() {
   super.destroy();
   conn.close();
}
```

Example: Calendar Servlet

- See the code for the *CalendarServlet* in the handout

```
public class CalendarServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {}
    public void getAppointments() throws IOException {}
    public void newAppointmentForm() throws IOException {}
    public void insertNewAppointment() throws IOEXception {}
}
```