More Servlets

Advanced Topics in Java

Khalid Azim Mughal
khalid@ii.uib.no
http://www.ii.uib.no/~khalid/atij/

Version date: 2006-09-04

Overview

- Redirecting Requests
- Sending Status Codes
- Sharing Data and Scopes: Session, Context and Request
- Session Management: `HttpSession`
  - Session Support Using Cookies
- Example: Implementing a Multi-servlet Web Application
- Servlets and Threads
- Multi-Thread and Single-Thread Models
- URL Pattern Mapping
  - Default and Custom URLs
- Servlet Development and Debugging: Handling Exceptions
- Secure Servlets: FORM-based Authentication
- More Cookies
- Logging
- Creating WARs

Redirecting Requests

- If a servlet for some reason cannot handle the user request it can *redirect* the request.
- The `HttpServletResponse` interface provides the following method to redirect requests:

`void sendRedirect(String location) throws IOException`	Sends a temporary redirect response using the specified redirect location URL. The URL can be a relative URL which the servlet container converts to a absolute URL. If the response has already been committed, this method throws an `IllegalStateException`. After using this method, the response should be considered to be committed and should *not* be modified.

- The browser *on receiving the redirection* will automatically retrieve the resource denoted by the URL.
  – Note the redirection is exposed to the client in the response. It is not transparent.
- In order to redirect, the response should not already have been committed.
  If so, a `IllegalStateException` will be thrown.
- See `SimpleHoroServletWithRedirection.java`.

Sending HTTP Status Codes

- The `HttpServletResponse` interface provides constants for the HTTP status codes:

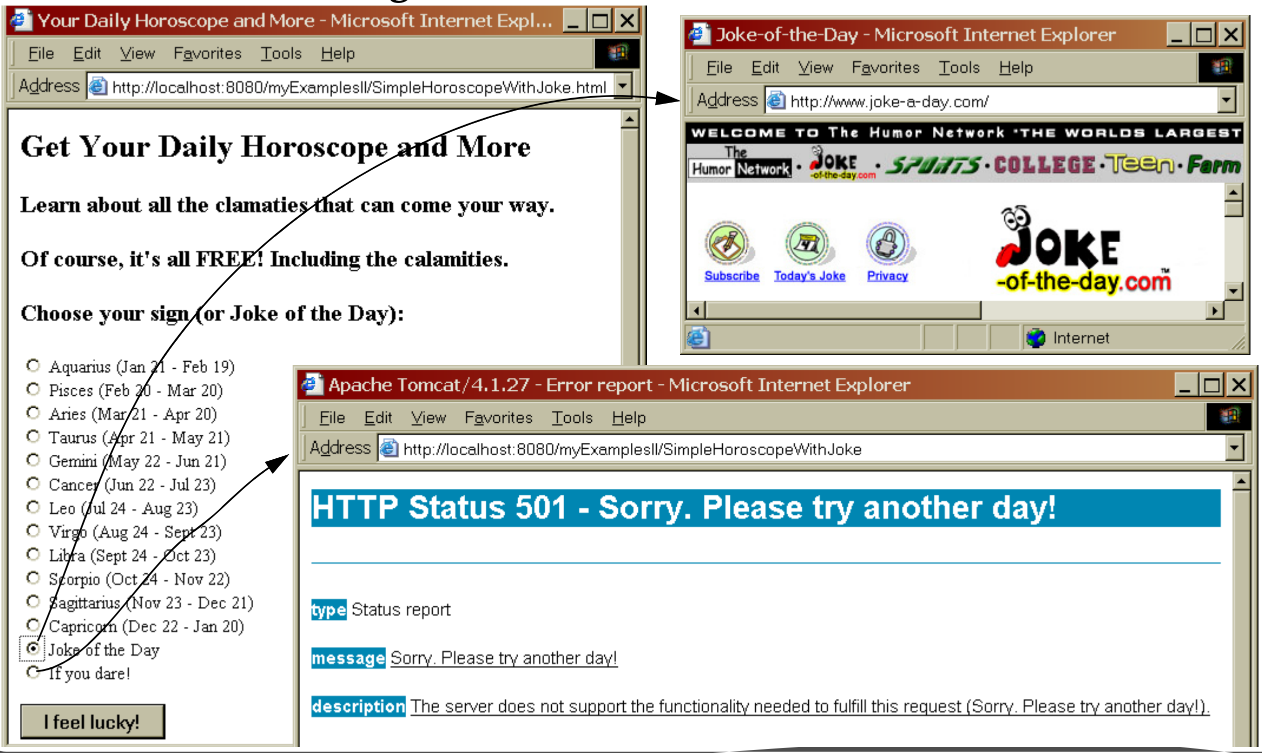
```
HttpServletResponse.SC_OK              // HTTP Status-Code 200: OK
HttpServletResponse.SC_NOT_FOUND       // HTTP Status-Code 404: Not Found
HttpServletResponse.SC_NOT_IMPLEMENTED // HTTP Status-Code 501: Not Implemented
```

- The `HttpServletResponse` interface provides the following methods for a servlet to send HTTP status codes as a response:

`void sendError(int sc) throws IOException` `void sendError(int sc, String msg) throws IOException`	Sends the specified HTTP status code as response. If the response has already been committed, this method throws an `IllegalStateException`. After using this method, the response should be considered to be committed and should *not* be modified.

- Normally the browser creates a server error page with appropriate message using the status code.

Using Redirection and Status Codes



Your Daily Horoscope and More - Microsoft Internet Expl...

File Edit View Favorites Tools Help

Address http://localhost:8080/myExamplesII/SimpleHoroscopeWithJoke.html

Get Your Daily Horoscope and More

Learn about all the clamaties that can come your way.

Of course, it's all FREE! Including the calamities.

Choose your sign (or Joke of the Day):

- ○ Aquarius (Jan 21 - Feb 19)
- ○ Pisces (Feb 20 - Mar 20)
- ○ Aries (Mar 21 - Apr 20)
- ○ Taurus (Apr 21 - May 21)
- ○ Gemini (May 22 - Jun 21)
- ○ Cancer (Jun 22 - Jul 23)
- ○ Leo (Jul 24 - Aug 23)
- ○ Virgo (Aug 24 - Sept 23)
- ○ Libra (Sept 24 - Oct 23)
- ○ Scorpio (Oct 24 - Nov 22)
- ○ Sagittarius (Nov 23 - Dec 21)
- ○ Capricorn (Dec 22 - Jan 20)
- [✓ Joke of the Day]
- ○ If you dare!

I feel lucky!

Joke-of-the-Day - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://www.joke-a-day.com/

WELCOME TO The Humor Network 'THE WORLDS LARGEST

The Humor Network · JOKE -of-the-day.com · SPORTS · COLLEGE · Teen · Farm

Subscribe Today's Joke Privacy

JOKE -of-the-day.com

Internet

Apache Tomcat/4.1.27 - Error report - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Address http://localhost:8080/myExamplesII/SimpleHoroscopeWithJoke

HTTP Status 501 - Sorry. Please try another day!

type Status report

message Sorry. Please try another day!

description The server does not support the functionality needed to fulfill this request (Sorry. Please try another day!).

- Servlet source code: `SimpleHoroServletWithRedirection.java`.
- Shows use of indirection and status codes by `sendRedirect()` and `sendError()` methods, respectively.

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                   throws ServletException, IOException {
    // Get the sign.
    String sign = request.getParameter("sign").toLowerCase();
    // Check if redirection is necessary.
    if (sign.equals("joke")) {
        response.sendRedirect("http://www.comedycentral.com/jokes/index.jhtml");
        return; // No further processing

    // Send status code, if this choice is not implemented.
    if (sign.equals("dare")) {
        response.sendError(
             HttpServletResponse.SC_NOT_IMPLEMENTED,
             "Sorry. Please try another day!");
        return; // No further processing
    }
    ...
}
```

Sharing Data in a Web Application

- In order to share data in a web application, the following *scopes* can be utilized depending on the business logic:
  - *Session Scope* (`HttpSession`) defines data which is visible only in a session associated with a particular client.
  - *Context Scope* (`ServletContext`) defines data which is visible to any client during the life time of the web application.
  - *Request Scope* (`HttpServletRequest`) defines data which is visible only as along as the request is being serviced.
- Examples:
  - If items in a shopping cart for a client should only be visible in the session associated with the client and not in any other sessions, then these items can be handled using session scope.
  - If business logic requires that a list of users for a web application should be visible to all servlets in the application for authentication purposes, then the list can reside in the servlet context associated with the application, i.e. it will have context scope.
  - If it is desired that specific information about a particular item is only valid during servicing of a request for this item and should not be available in requests for other items, then this information can be stored in the request, i.e. it will have request scope.

Sharing Data using Attributes

- In each of the three scopes (session, context, request), information can be stored and retrieved using *attributes*. An attribute has a *name* and a *value*, and defines an *entry*: `<String attributeName, Object attributeValue>`.
- Each scope (`HttpSession`, `ServletContext`, `HttpServletRequest`) provides the following methods to store and retrieve attributes:

`void setAttribute(String name, Object value)`	Binds an object to this scope, using the name specified, i.e. the scope stores the entry `<name, value>`. If an object of the same name is already bound to the scope, the old object is replaced.
`Object removeAttribute(String name)`	Removes the object bound with the specified name from this scope. If the scope does not have an object bound with the specified name, this method does nothing.
`Object getAttribute(String name)`	Returns the object bound with the specified name in this scope, or `null` if no object is bound under the name.
`Enumeration getAttributeNames()`	Returns an `Enumeration` of `String` objects containing the names of all the objects bound to this scope.

Attribute Scope Creation and Termination

- From the view point of the servlet:

Scope:	Creation	Termination
Session	When the `getSession()` method is called by the servlet.	When the `invalidate()` method is called by the servlet or the session is timed out by the server.
Context	When the application is loaded.	When the application is terminated.
Request	When the `service()` method is invoked on the servlet with the request object as parameter.	When the `service()` method has terminated, i.e. the servlet has handled the request.

Sessions

- The HTTP protocol is *memoryless*, i.e. it is *stateless*.
  The server cannot remember information from one request to the next from a client.
- A *session* provides the means for tracking the interaction between a particular client and the web application.
  - A session is a repository of all pertinent information about a sequence of continuous requests and responses between the server and a particular client.
- A session scenario proceeds as follows:
  - A request from a client results in the creation of a session on the sever-side.
    The server also assigns a unique *session ID* which identifies the session.
    This session ID is included in all responses to the client.
  - Any subsequent requests from this client contain the session ID which the server uses to identify the session associated with this client.
  - The session terminates either if a client action ends the session or the server detects a period of inactivity of predefined length on the part of the client.
  - After session termination, all interaction information about the user and the session is expunged.

The `HttpSession` Interface

- A session implements the `javax.servlet.http.HttpSession` interface.
- A session is obtained by invoking one of the following methods on a `HttpServletRequest` object:

`HttpSession getSession(boolean instantiate)`	Returns the current *session* associated with the request if there is one. Otherwise creates one if argument `instantiate` is `true`.
`HttpSession getSession()`	This method is equivalent to `getSession(true)`.

- The association of a client to a particular session is done transparently by the server. No extra code is necessary apart from calling one of the above methods.

Using the Session as a Repository: Session Scope

- When a client issues a request, the session for this client is identified by the server and the attributes in this session are available for handling the request.
  - A session is thus accessible to the thread servicing the request.
  - We say that the attributes have *session scope*, i.e. they are visible to any thread servicing a request that belongs to the session.
  - Session scope implies that the session is available to any servlet executed by the thread as part of handling the request from the client associated with the session.
- Typical example of using a session is a shopping cart.
  - The business logic of a shopping cart can be implemented using a session.
  - A session is created when the client starts shopping and expunged when the order is finally placed.

Functionality of the `HttpSession` Interface

- The following method can be used to obtain various information about a session:

`boolean isNew()`	Returns `true` if the client does not yet know about the session or if the client chooses not to join the session.
`String getId()`	Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.
`long getCreationTime()`	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
`long getLastAccessedTime()`	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
`void invalidate()`	Invalidates this session, then unbinds any objects bound to it.

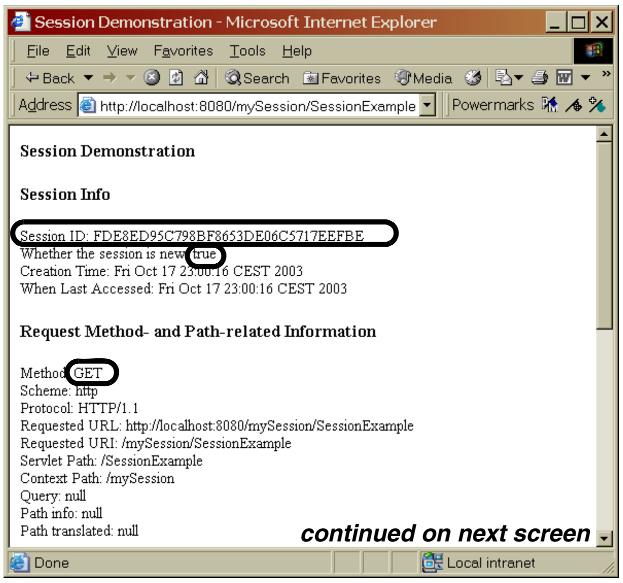
- The following methods can be used to set and get the *time-out value* of a session:

`int getMaxInactiveInterval()`	Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
`void setMaxInactiveInterval( int interval)`	Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never time-out.

Demonstrating Session Scope

- Servlet `SessionExample` presents a form to register attributes, i.e. <*name, value*> entries.
- The first request to the servlet creates a session.
- Clicking the `Add` button adds the attribute to the current session, and also lists all the attributes in this session.
- Clicking the `Clear` button invalidates the current session. A new session is not created until the user registers an attribute.

New Session (Start Screen)


Session Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/mySession/SessionExample
Powermarks
Done
Local intranet

Session Demonstration

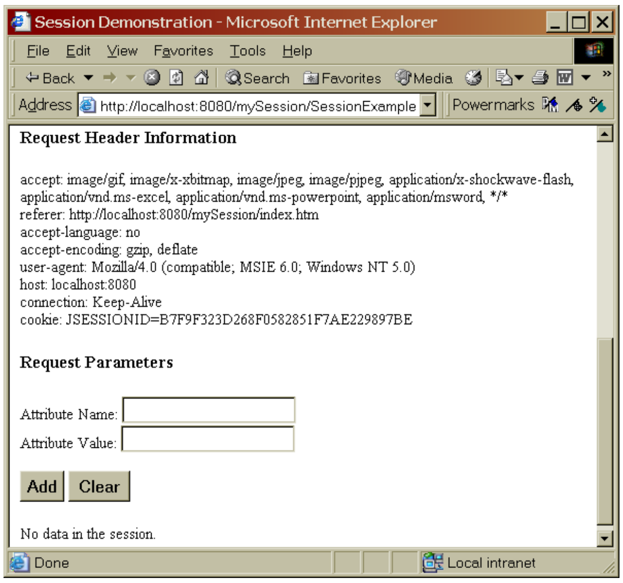
Session Info

Session ID: FDE8ED95C798BF8653DE06C5717EEFBE
Whether the session is new: true
Creation Time: Fri Oct 17 23:00:16 CEST 2003
When Last Accessed: Fri Oct 17 23:00:16 CEST 2003

Request Method- and Path-related Information

Method: GET
Scheme: http
Protocol: HTTP/1.1
Requested URL: http://localhost:8080/mySession/SessionExample
Requested URI: /mySession/SessionExample
Servlet Path: /SessionExample
Context Path: /mySession
Query: null
Path info: null
Path translated: null

continued on next screen


Session Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/mySession/SessionExample
Powermarks
Add
Clear
Done
Local intranet

Request Header Information

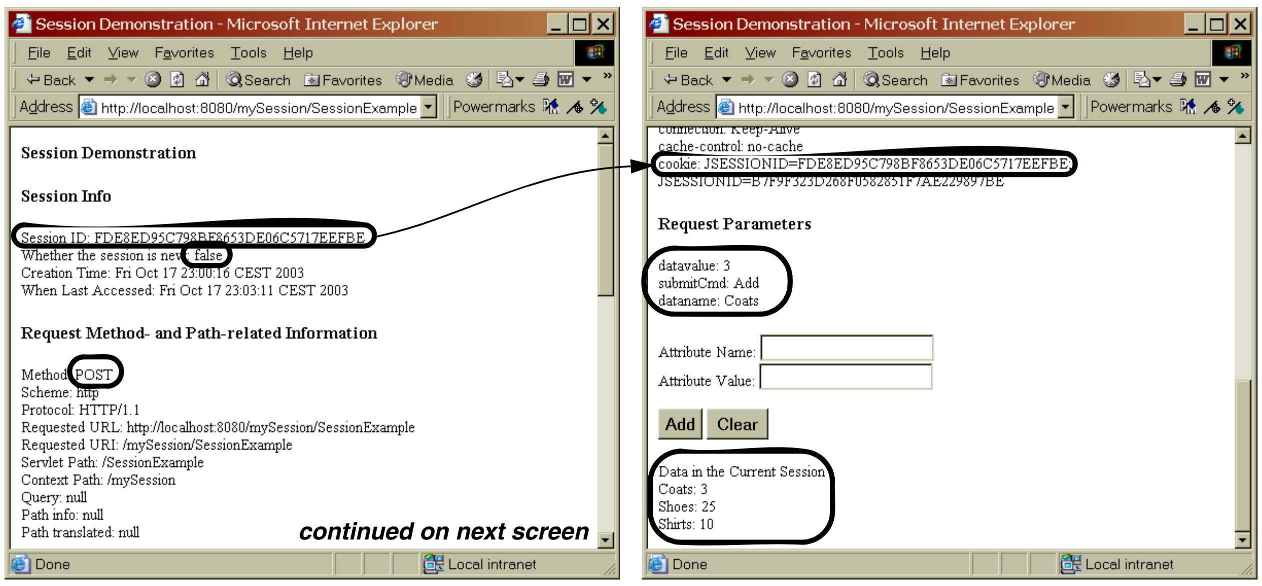
accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
referer: http://localhost:8080/mySession/index.htm
accept-language: no
accept-encoding: gzip, deflate
user-agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
host: localhost:8080
connection: Keep-Alive
cookie: JSESSIONID=B7F9F323D268F0582851F7AE229897BE

Request Parameters

Attribute Name:
Attribute Value:

No data in the session.

"Joined" Session: Subsequent Requests


Session Demonstration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/mySession/SessionExample
Powermarks
Done
Local intranet

Session Demonstration

Session Info

Session ID: FDE8ED95C798BF8653DE06C5717EEFBE
Whether the session is new: false
Creation Time: Fri Oct 17 23:00:16 CEST 2003
When Last Accessed: Fri Oct 17 23:03:11 CEST 2003

Request Method- and Path-related Information

Method: POST
Scheme: http
Protocol: HTTP/1.1
Requested URL: http://localhost:8080/mySession/SessionExample
Requested URI: /mySession/SessionExample
Servlet Path: /SessionExample
Context Path: /mySession
Query: null
Path info: null
Path translated: null

continued on next screen

connection: Keep-Alive
cache-control: no-cache
cookie: JSESSIONID=FDE8ED95C798BF8653DE06C5717EEFBE;
JSESSIONID=B7F9F323D268F0582851F7AE229897BE

Request Parameters

datavalue: 3
submitCmd: Add
dataname: Coats

Attribute Name:
Attribute Value:

Add Clear

Data in the Current Session
Coats: 3
Shoes: 25
Shirts: 10

Session Support Using Cookies

- The servlet `SessionExample` shows how sessions are supported using cookies.
- The server container sends the session ID to the client in a `Set-Cookie` header of the response:
  `set-cookie: JSESSIONID=B0A1814A4AE32F7F1D7963FCD590542B;...`
- The client sends the session ID in subsequent requests to the server in the `Cookie` header of a request:
  `cookie: JSESSIONID=B0A1814A4AE32F7F1D7963FCD590542B;...`
- The session ID, given by the name `JSESSIONID`, is used by the servlet container to retrieve and associate the correct session with the client.
- Note that session identification and association is transparent to the servlet.
- Also note that cookies must be *enabled* in the browser, i.e. that the browser is willing to accept cookies.
- In case cookies are not enabled in the browser, alternate methods, such as *URL rewriting*, can be considered.

The States of a Session: New, Joined and Invalidated

- See `SessionExample.java`.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                   throws IOException, ServletException
{   ...
    HttpSession session = request.getSession();
    ...
    if (command == null) // Sufficient to check for null.
                         // Not sufficient to just call session.isNew().
        // List the data in the session.
        listSessionData(session, out); // Should be empty.
    else if (command.equalsIgnoreCase("Add")) { // Subsequent requests (joined).
        if (!dataName.equals("") && !dataValue.equals(""))
            // Add parameters to the session.
            session.setAttribute(dataName, dataValue);
        // List the data in the session.
        listSessionData(session, out);
    } else if (command.equalsIgnoreCase("Clear"))
        session.invalidate();                  // Invalidated.
    ...
}
```

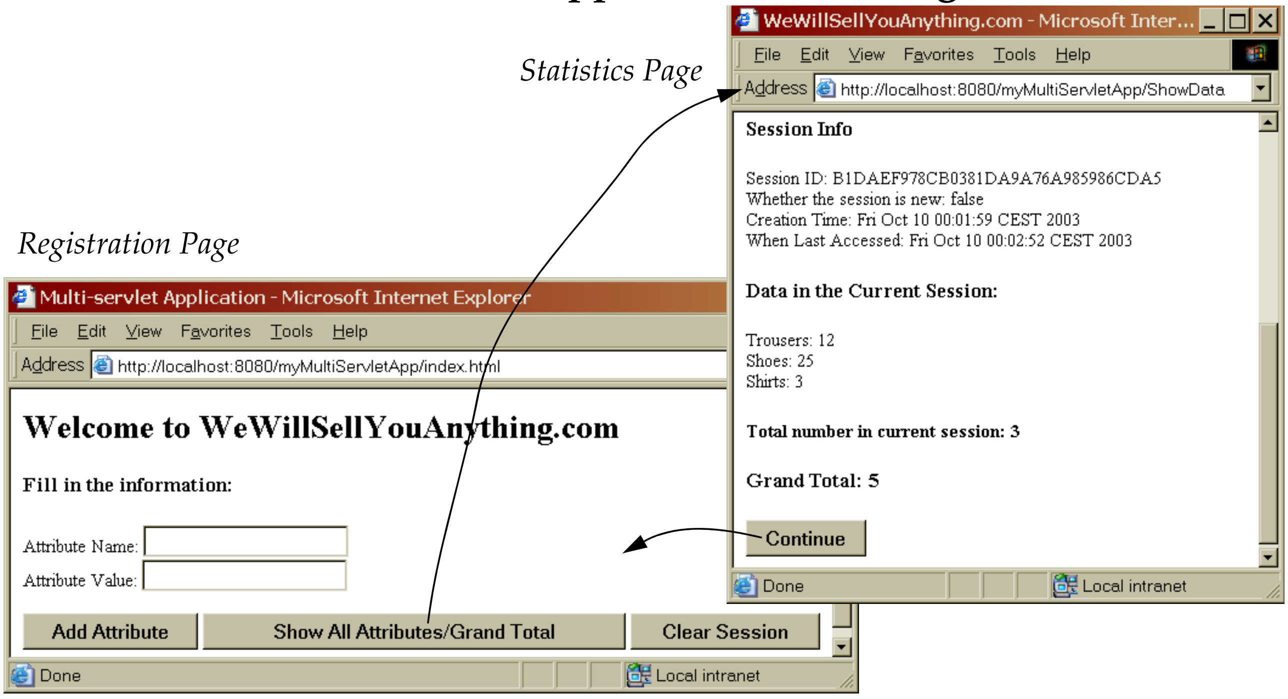
Example of a Multi-servlet Web Application

- Document root of the application: `myMultiServletApp`.
- The application allows the client to add attributes to the current session (the Add Attribute button).
- It lists the attributes in the current session, together with the grand total of all attributes that were ever added in all sessions up to present time (the Show All Attributes/Grand Total button).
  - The statistics page allows the client to navigate to the registration page (the Continue button).
- The client can terminate the current session (the Clear Session button).

Multi-servlet Application: Main Pages

Statistics Page

Registration Page



Multi-servlet Application - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/myMultiServletApp/index.html
Add Attribute
Show All Attributes/Grand Total
Clear Session
Done
Local intranet
WeWillSellYouAnything.com - Microsoft Inter...
File Edit View Favorites Tools Help
Address http://localhost:8080/myMultiServletApp/ShowData
Continue
Done
Local intranet

Welcome to WeWillSellYouAnything.com

Fill in the information:

Attribute Name:

Attribute Value:

Session Info

Session ID: B1DAEF978CB0381DA9A76A985986CDA5
Whether the session is new: false
Creation Time: Fri Oct 10 00:01:59 CEST 2003
When Last Accessed: Fri Oct 10 00:02:52 CEST 2003

Data in the Current Session:

Trousers: 12
Shoes: 25
Shirts: 3

Total number in current session: 3

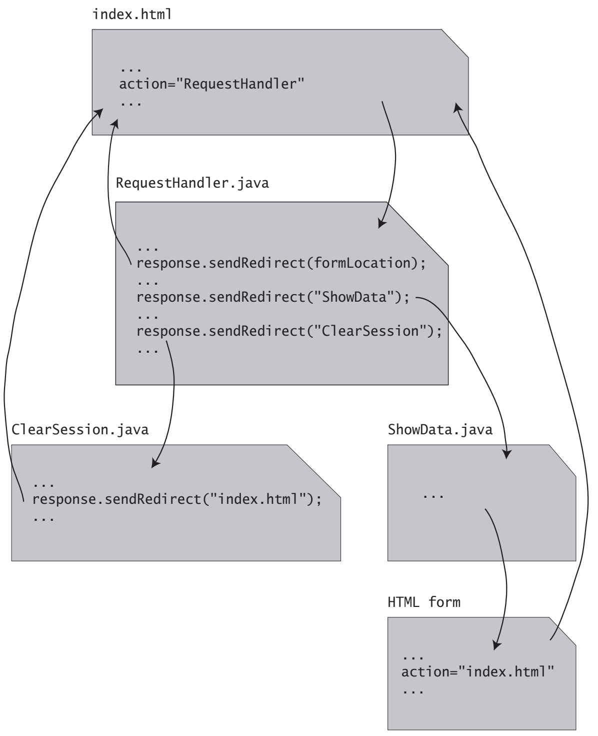
Grand Total: 5

Multi-servlet Application: Implementation

Multi-servlet Application Structure

- The application consists of three servlets (`RequestHandler.java`, `ShowData.java`, `ClearSession.java`) and one HTML document (`index.html`).
- The implementation demonstrates following aspects:
  - HTML form created dynamically by a servlet (`ShowData.java`) and also retrieved from a HTML file (`index.html`).
  - demonstrates session, context and request scopes.
  - demonstrates redirection of requests.
- See the figure on the next page for an overview of control flow in the application.

Multi-servlet Application: Control Flow


index.html
...
action="RequestHandler"
...
RequestHandler.java
...
response.sendRedirect(formLocation);
...
response.sendRedirect("ShowData");
...
response.sendRedirect("ClearSession");
...
ClearSession.java
...
response.sendRedirect("index.html");
...
ShowData.java
...
HTML form
...
action="index.html"
...

The Registration Page

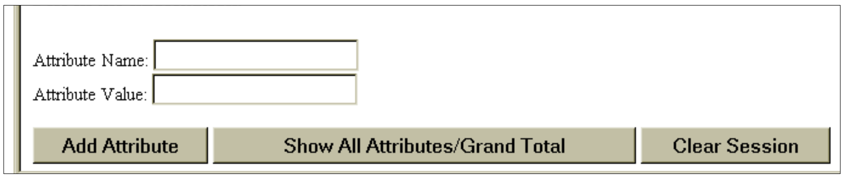
- The HTML form defines the main page of the application (file `index.html`).

Servlet called *Using POST method*

```
<form action="RequestHandler" method="post">
    Attribute Name:
    <input type="text" size="20" name="dataname"/><br/>
    Attribute Value:
    <input type="text" size="20" name="datavalue"/>
    <br><br>
    <input type="submit" name="submitCmd" value="Add Attribute"/>
    <input type="submit" name="submitCmd"
                       value="Show All Attributes/Grand Total"/>
    <input type="submit" name="submitCmdv value="Clear Session">
</form>
```

Defines two text fields

Defines three commands



The Main Servlet: `RequestHandler`

- The `RequestHandler` servlet creates a new session if necessary (`doPost()` method).

```
HttpSession session = request.getSession();
```

- The `RequestHandler` servlet reads the HTML form parameters (`doPost()` method).

```
String dataName  = request.getParameter("dataname");
String dataValue = request.getParameter("datavalue");
String command   = request.getParameter("submitCmd");
```

- The RequestHandler servlet interprets the HTML form parameters (doPost() method).

```
String formLocation = "index.html";
if (command == null)                            // (1)
    // Redirect to the HTML form.
    response.sendRedirect(formLocation);
else if (command.startsWith("Add")) {           // (2)
    if (!dataName.equals("") && !dataValue.equals("")) {
        // Add parameters to the session.
        session.setAttribute(dataName, dataValue);
        // Update grand total.
        updateGrandTotal();
       }
    // Redirect to the HTML form.
    response.sendRedirect(formLocation);
} else if (command.startsWith("Show"))         // (3)
    // Redirect to the ShowData servlet.
    response.sendRedirect("ShowData");
else if (command.startsWith("Clear"))          // (4)
    // Redirect to the ClearSession servlet.
    response.sendRedirect("ClearSession");
```

- The RequestHandler servlet stores "global" information (i.e. the company name) in the servlet context so that other servlets can also share this information.
  – This information is set only once in the init() method when the servlet is loaded.

```
public void init()
{
    // Set the company name in the servlet context.
    ServletContext context = getServletContext();
    context.setAttribute("companyName",
                         "WeWillSellYouAnything.com");
}
```

- The `RequestHandler` servlet updates the attribute count (i.e. the grand total) in the servlet context so that other servlets can also share this information.
  - The update is made thread-safe by synchronizing on the servlet context.

```
private void updateGrandTotal()
{
    ServletContext context = getServletContext();
    synchronized(context) { // Drastic but thread-safe.
        String attributeName = "grandTotal";
        String attributeValue =
               (String) context.getAttribute(attributeName);
        if (attributeValue != null) {  // Update
           context.setAttribute(attributeName,
                 String.valueOf(Integer.parseInt(attributeValue) +
                                1));
        } else  // First time.
           context.setAttribute(attributeName, "1");
    }
}
```

The Statistics Page: `ShowData` Servlet

- The `doPost()` method of the `ShowData` servlet creates a response, including appropriate information.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                   throws IOException, ServletException
{   ...
   String title = getCompanyName(); // Get the name from the servlet context.
    ...
   // Get the session, if there is one. Does not create one if there is none.
    HttpSession session = request.getSession(false);
    if (session != null) {
        ServletUtil.echoSessionInfo(session, out);
        // List data in the session.
        listSessionData(session, out);
    }
    // Send grand total
    sendGrandTotal(out);
    // Render the form.
    renderForm(response, out);
    ...
}
```

- The ShowData servlet retrieves the *attributes from the current session* and includes it in the response.

```
private void listSessionData(HttpSession session, PrintWriter out)
                             throws IOException, ServletException
{       int total = 0; String attributes ="";
        Enumeration names = session.getAttributeNames();
        while (names.hasMoreElements()) {
            String name = (String) names.nextElement();
            String value = session.getAttribute(name).toString();
            attributes += (name + ": " + value + "<br/>");
            total++;
        }
        if (total == 0)
            out.println("<h3>No data in the current session.</h3>");
        else {
            out.println("<h3>Data in the Current Session:</h3>");
            out.println("<p>" + attributes + "</p>");
            out.println("<h4>Total number in current session: " +
                        total + "</h3>");
        }
}
```

- The ShowData servlet retrieves the *company name from the servlet context* and includes it in the response.

```
private String getCompanyName()
{
    ServletContext context = getServletContext();
    String attributeName = "companyName";
    return (String) context.getAttribute(attributeName);
}
```

- The ShowData servlet retrieves *the grand attribute total from the servlet context* and includes it in the response.

```
private void sendGrandTotal(PrintWriter out)
{   ServletContext context = getServletContext();
    String attributeName = "grandTotal";
    String attributeValue =
           (String) context.getAttribute(attributeName); // Cast necessary
    String total = "0";
    if (attributeValue != null)
        total = attributeValue;

    out.println("<h3>Grand Total: " + total + "</h3>");
}
```

- The ShowData servlet generates a HTML form (as part of the response) to allow the client to continue with attribute registration.

```
private void renderForm(HttpServletResponse response,
                        PrintWriter out)
                  throws IOException, ServletException
{
    out.println("<p>");
    out.print("<form action=\"index.html\""); // Go to form
    out.println("method=\"get\">");
    out.println(
             "<input type=\"submit\" name=\"command\" value=\"Continue\"/>");
    out.println("</form>");
    out.println("</p>");
}
```

Session Invalidation: ClearSession Servlet

- The ClearSession servlet invalidates the current session, if one exists.
  - It does not creates a new session.

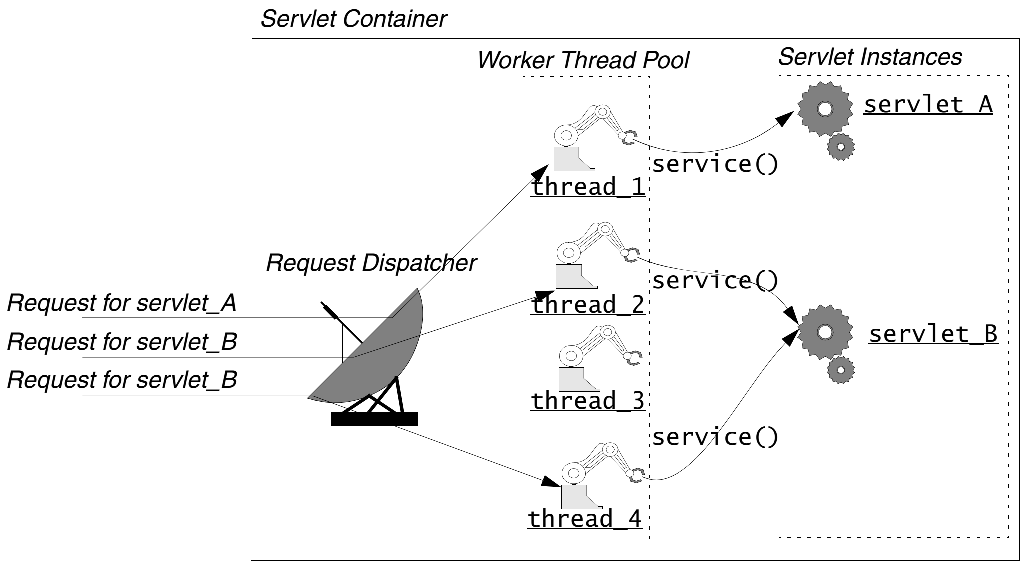
```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                   throws IOException, ServletException
{
    // Get the current session, if there is one.
    HttpSession session = request.getSession(false);
    // Invalidate session.
    if (session != null)
        session.invalidate();
    // Redirect to the HTML form.
    response.sendRedirect("index.html");
}
```

More Servlet Utilities: `ServletUtil`

- The static method `echoSessionInfo()` includes pertinent information about the current session in a response:

```
public static void echoSessionInfo(HttpSession session,
                                   PrintWriter out)
                           throws ServletException, IOException {
    out.println("<h3>Session Info</h3>");
    out.println("<p>");
    out.println("Session ID: "+ session.getId()  + "<br/>");
    out.println("Whether the session is new: "+ session.isNew() +
                "<br/>");
    out.println("Creation Time: "+
              new Date(session.getCreationTime()) + "<br/>");
    out.println("When Last Accessed: " +
              new Date(session.getLastAccessedTime()) + "<br/>");
    out.println("</p>");
}
```

Multi-thread Servlet Model



Only one instance of a servlet, and several requests can execute concurrently in this servlet instance.

Remarks on Multi-thread Servlet Model

- The multi-thread servlet model is the default mode of execution.
- As is the case with any Java application in a multi-thread environment, the servlet must take the necessary steps to ensure data integrity.
- Fields in a servlet are not thread-safe, therefore access to them must be synchronized.
  – However, local variables are thread-safe as a new copy is created on each method invocation in a thread.
- Attributes in a context are not thread-safe, as any number of threads (requests) can be accessing them.
- Attributes in a session are also not thread-safe, as a client can send simultaneous requests from different browsers, thus accessing the same session.
- Access to shared data in a session or a context must be synchronized either on the data or on the session/context.
- Attributes in a request are always thread-safe as the request object passed to the `service()` method is isolated from other request objects in other invocations of the `service()` method.
  – It is not a good idea to "cache" a request object.

Thread-safety and Variables Summary

Variables	**In Multi-thread Servlet Model**
Local variables	Thread-safe
Instance variables	Not thread-safe
Static variables	Not thread-safe

Thread-safety and Attribute Scope Summary

Scope	**In Multi-thread Servlet Model**	**Solution**
Context	Not thread-safe	Synchronize on shared data in the context.
Session	Not thread-safe	Synchronize on shared data and/ or on the session.
Request	Thread-safe	OK

The `/servlet/` Pattern in URLs to invoke Servlets

- A servlet container has a default servlet with the registered name `invoker`. The `/servlet/` pattern in a URL maps to this servlet which invokes the real servlet designated in the URL.

Examples of using the `/servlet/` pattern in URLs.

Assume that the following `servlet` element has been specified in the `web.xml` file of the web application:

```
<servlet>
    <servlet-name>mySimpleServlet</servlet-name>
    <servlet-class>StopTheWorld</servlet-class>
</servlet>
```

- *Default URL* of the servlet can be used to invoke the servlet (i.e. the class name specified in `<servlet-class>` element):
  `http://localhost:8080/myExamples/servlet/StopTheWorld`
- *Registered name* of the servlet can be used to invoke the servlet (i.e. the servlet name specified in `<servlet-name>` element):
  `http://localhost:8080/myExamples/servlet/mySimpleServlet`
- Using the `/servlet/` pattern does *not* require a `servlet-mapping` in the `web.xml` file of the web application.

Note that the servlet can *always* be invoked using *customized URLs* based on the URL pattern specified in the `servlet-mapping` element in the `web.xml` file:

```
<servlet-mapping>
    <servlet-name>mySimpleServlet</servlet-name>
    <url-pattern>/SimpleServlet/*</url-pattern>
</servlet-mapping>
```

- *Customized URLs* of the servlet (in `<url-pattern>` element) can be used to invoke the servlet:
  `http://localhost:8080/myExamples/SimpleServlet`
  `http://localhost:8080/myExamples/SimpleServlet/`
  `http://localhost:8080/myExamples/SimpleServlet/more`
- The `/servlet/` pattern with either the default URL or the registered name *only* works in invoking the servlet if the `invoker` servlet is *enabled* in the servlet container.
- Customized URLs work as long as an appropriate `<servlet-mapping>` element is specified, and do not depend on whether the `invoker` servlet is enabled or not.
- The `invoker` servlet of the servlet container can be disabled by commenting out the following entry in the *<tomcat-home>*`/conf/web.xml` file:

```
<servlet-mapping>
    <servlet-name>invoker</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>
```

Logging

- Printing messages or debugging information on the console using `System.out` or `System.err` is not always the best solution.
  - This information is not persistent unless captured somehow, and is certainly not recommended in a production environment.
- Logging is a better solution, which can also, for example, be used for monitoring the performance of web applications.
- The information is written to a file (called the *servlet log file*) whose name and type is specific to the servlet container.
  - Log files are usually created in the <*tom-cat home directory*>`/logs` directory.
  - For example, Tomcat logs information about servlet deployment in log files.
  - Configuration of log files for different web applications can be done by providing specific information in the <*tom-cat home directory*>`/conf/server.xml` file.
- The abstract `GenericServlet` class and the `ServletContext` interface define the following two methods for logging.

`void log(String msg)`	*Logs* the specified message to the *servlet log file*.
`void log(String msg, Throwable exception)`	*Logs* the specified message and a stack trace to the *servlet log file*.

  - The methods in the `GenericServlet` class prepend the servlet name to the logged information.
- Example: Logging commands in the `RequestHandler` servlet.

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
                   throws IOException, ServletException
{   ...
    String command   = request.getParameter("submitCmd");
    ...
    log("command: " + command);
    ...
}
```

Contents of log file `localhost_log.2003-10-12.txt`:

```
...
2003-10-12 11:50:01 RequestHandler: command: Add Attribute
2003-10-12 11:50:04 RequestHandler: command: Add Attribute
2003-10-12 11:50:08 RequestHandler: command: Add Attribute
2003-10-12 11:50:09 RequestHandler: command: Show All Attributes/Grand Total
...
```

Handling Exceptions

- Two common scenarios where an exception is thrown and caught in a catch block.
  - In the catch block, the action is to send an error message as response:

```
try { ... }
catch (SomeException se) {
    response.sendError(APPROPRIATE_HTTP_CODE, "explanation");
    // Note that the response has now been committed.
}
```

  - In the catch block, the action is to log the exception:

```
try { ... }
catch (SomeException se) {
    log("Exception occurred", se); // stack trace will be logged.
}
```

Defining Exceptions

- Overriding the `do`*HttpRequestMethodName*`()` methods does *not* allow new *checked* exception types to be specified in the `throws` clause unless these checked exception types are subclasses of `ServletException` or `IOException`.

```
class MajorIOException extends IOException {
    MajorIOException(String message) { super(message); }
}

public class ClearSession extends HttpServlet {

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws MajorIOException
    { ...
      throw new MajorIOException("Serious I/O Problem");
      ...
    }
}
```

- There are more sophisticated mechanisms for handling exceptions *declaratively*, but we will not discuss them here.

More Cookies

- Some typical uses of cookies:
  - Identification of a client during a session
  - Remembering username and password
  - Customizing responses, for example, to provide focused advertising
- Some Problems with Cookies
  - Cookies can be a privacy problem, not a security problem.
  - Some privacy problems:
    - actions from previous session can be remembered
    - personal information can be linked to previous actions
    - information can be shared with others through willing third party
    - stored sensitive information can present a problem

- Creating and Adding Cookies to the Response
  - The `addCookie()` method of the `HttpServletResponse` interface is used to add a cookie to the response.
  - An appropriate HTTP `Set-Cookie` response header is created.
  - *Note that the cookie has to be added to the response each time if the client is to return it in the next request.*

```
Cookie uidCookie = new Cookie("user", "uid999"); // Create a Cookie.
uidCookie.setMaxAge(60*60*24*365); // 1 year
response.addCookie(uidCookie);
```

  - Before adding a cookie to the response, various characteristics of the cookie can be tailored using `set`*Attribute*`()` methods, where *Attribute* is the name of the attribute you want to specify. There are also corresponding `get`*Attribute*`()` methods to retrieve the attribute value.

`void setComment(String purpose)`	Specifies a comment that describes a cookie's purpose.
`void setDomain(String pattern)`	Specifies the domain within which this cookie should be presented.
`void setMaxAge(int expiry)`	Sets the maximum age of the cookie in seconds.

`void setPath(String uri)`	Specifies a path for the cookie to which the client should return the cookie.
`void setSecure(boolean flag)`	Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.
`void setValue(String newValue)`	Assigns a new value to a cookie after the cookie is created.
`void setVersion(int v)`	Sets the version of the cookie protocol this cookie complies with.

- For new cookies, the name is supplied in the constructor call. There is no `setName()` method. For cookies in the response, the `getName()` method can be used (see below).
- For new cookies, the value is supplied in the constructor call. For cookies from the client, the `getValue()` method can be used to extract the value from the cookie (see below).

- Reading Cookies from the Request
  - The client sends back cookies in each request.
  - The cookies can be obtained from the request by calling the `getCookies()` method of the `HttpServletRequest` interface. The method returns an array of `Cookie` objects corresponding to the cookies in the request. If there are no cookies in the request, the method returns `null`.

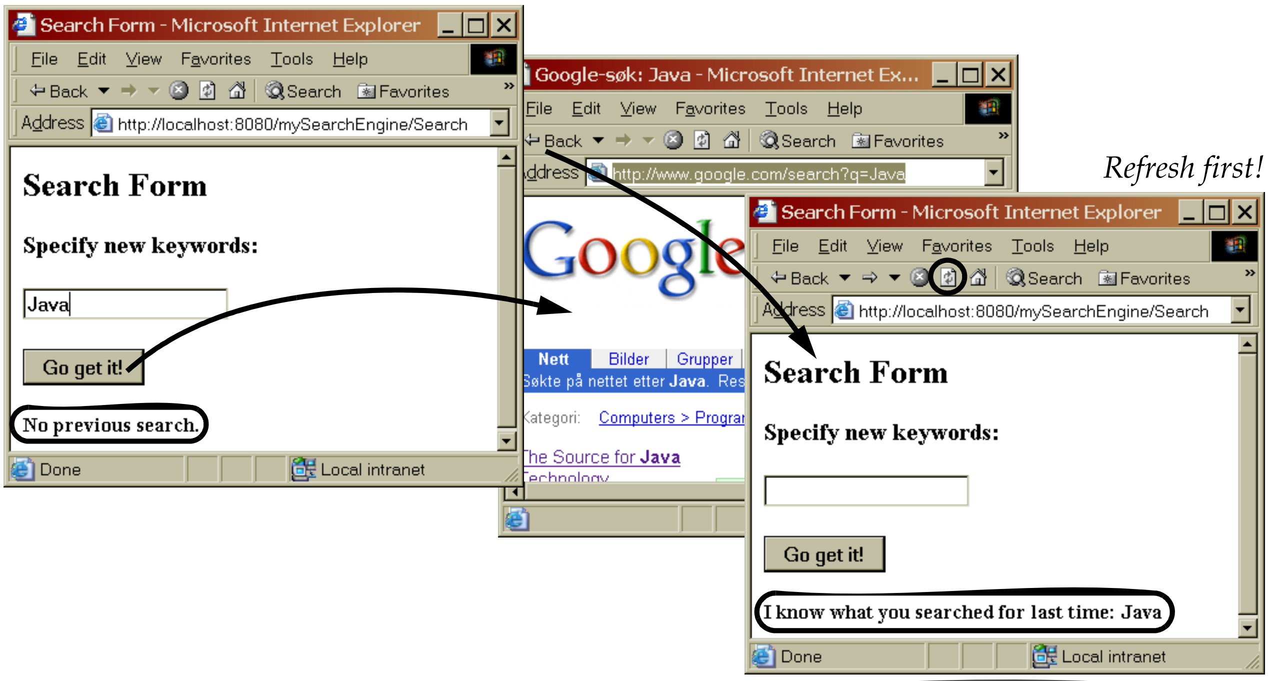
```
Cookie[] cookies = request.getCookies(); // Retrieve all cookies from request
```

  - Looking up the value of a particular cookie can be done in a loop:

```
String cookieValue;
if (cookies != null) {
  for(int i = 0; i < cookies.length; i++) {
    Cookie cookie      = cookies[i];
    String cookieName  = cookie.getName();
    if (cookieName.equals(requiredCookieName)) {
      cookieValue = cookie.getValue();
      break;
    }
  }
}
```

Example: Using Cookies

- The application submits a search to Google and remembers the last search.


Search Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/mySearchEngine/Search
Search Form
Specify new keywords:
Java
Go get it!
No previous search.
Done
Local intranet
Google-søk: Java - Microsoft Internet Ex...
Address http://www.google.com/search?q=Java
Google
Nett Bilder Grupper
Søkte på nettet etter Java.
Kategori: Computers > Progra
The Source for Java Technology
Refresh first!
Search Form - Microsoft Internet Explorer
Address http://localhost:8080/mySearchEngine/Search
Search Form
Specify new keywords:
Go get it!
I know what you searched for last time: Java
Done
Local intranet

- The servlet SearchEngineGUI creates the search form (doPost() method).

```
...
Cookie[] cookies = request.getCookies();
...
out.println(
    "<html><head><title>Search Form</title></head>\n" +
    "<head><title>Search Form</title></head>\n" +
    "<body>\n" +
    "<h1>Search Form</h1>\n" +
        "<form name=\"searchForm\"\n" +
              "action=\"SearchHandler\"\n" +
              "method=\"POST\">\n" +
            "<h2>Specify new keywords:</h2>\n" +
            "<p><input type=\"text\" name=\"searchCriteria\"/>"+
            "</p>\n" +
            "<p><input type=\"submit\" name=\"submitCmd\"" +
                 " value=\"Go get it!\"/>" +
            "</p>\n" +
        "</form>\n" +
             getPreviousSearchCriteria(cookies) +
   "</body>\n" +
   "</html>\n");
 ...
```

- The servlet `SearchEngineGUI` looks up the last search criteria.

```
  private String getPreviousSearchCriteria(Cookie[] cookies) {
    String cookieValue = "";
    if (cookies != null) {
      for(int i = 0; i < cookies.length; i++) {
        Cookie cookie      = cookies[i];
        String cookieName  = cookie.getName();
        if (cookieName.equals("previousSearchCriteria")) {
           cookieValue = cookie.getValue();
           break;
        }
      }
    }
    if (cookieValue.equals(""))
      return "<h3>No previous search.</h3>\n";
    else
      return "<h3>I know what you searched for last time: " +
              cookieValue + "</h3>\n";
  }
}
```

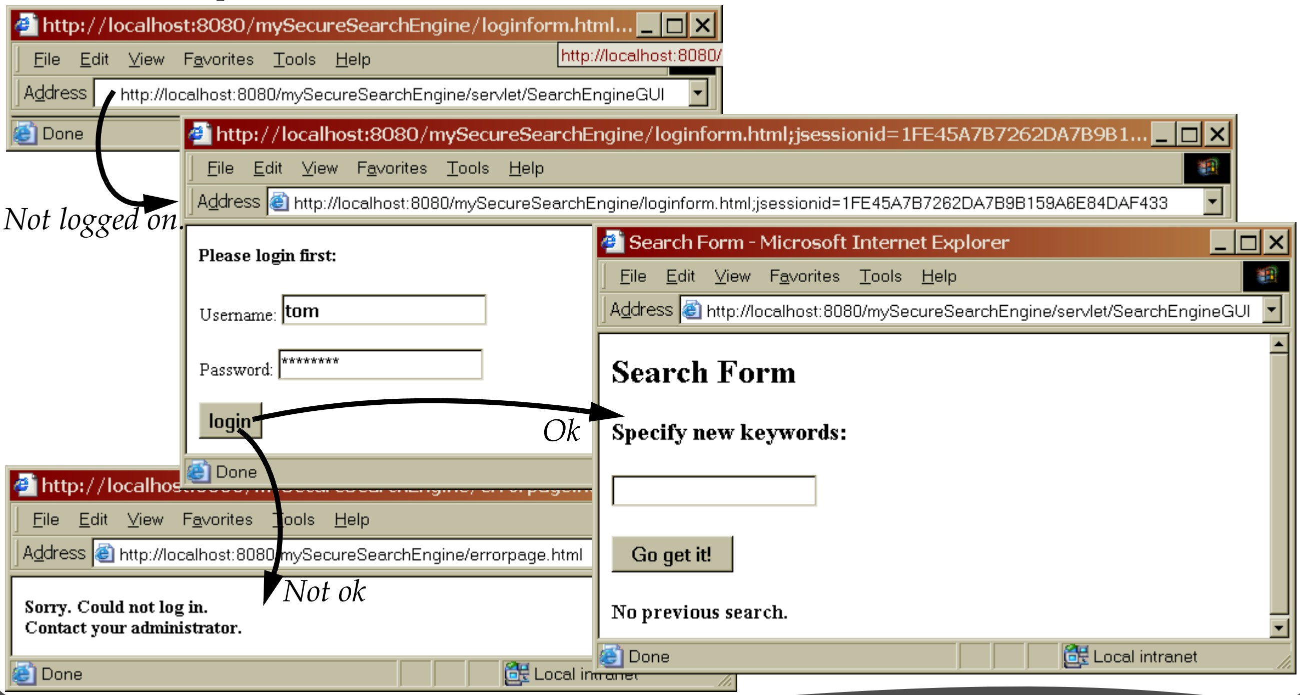
- The servlet `SearchHandler` creates the cookie to save the search criteria and redirects the request to Google (via the browser) (See the `doGet()` method).

```
    // Read the form parameter.
    String searchCriteria = request.getParameter("searchCriteria");
    // Check if current search is valid.
    if ((searchCriteria == null) ||
        (searchCriteria.length() == 0)) {
      response.sendError(HttpServletResponse.SC_NOT_FOUND,
                         "Missing search string.");
      return;
    }
    // Create a new cookie for this search.
    Cookie previousSearchCookie = new Cookie("previousSearchCriteria",
                                             searchCriteria);
    // Add it to the response.
    response.addCookie(previousSearchCookie);

    // Set up the uri and redirect.
    String uri = "http://www.google.com/" + "search?q=" +
                 URLEncoder.encode(searchCriteria, "UTF8");
    response.sendRedirect(uri);
```

Secure Servlets: FORM-based Authentication

- Example shows FORM-based authentication for a servlet (`mySecureSearchEngine`).


http://localhost:8080/mySecureSearchEngine/loginform.html...
File Edit View Favorites Tools Help
Address http://localhost:8080/mySecureSearchEngine/servlet/SearchEngineGUI
Done
Not logged on.
http://localhost:8080/mySecureSearchEngine/loginform.html;jsessionid=1FE45A7B7262DA7B9B1...
Address http://localhost:8080/mySecureSearchEngine/loginform.html;jsessionid=1FE45A7B7262DA7B9B159A6E84DAF433
Please login first:
Username: tom
Password: ********
login
Ok
Search Form - Microsoft Internet Explorer
Address http://localhost:8080/mySecureSearchEngine/servlet/SearchEngineGUI
Search Form
Specify new keywords:
Go get it!
No previous search.
Local intranet
Address http://localhost:8080/mySecureSearchEngine/errorpage.html
Not ok
Sorry. Could not log in.
Contact your administrator.

Procedure for FORM-based Authentication

1. Define users (their names, passwords and *roles*), who can access the servlet, in <*tomcat-home*>`\conf\tomcat-users.xml`:

```
<tomcat-users>
  ...
  <role rolename="role1"/>
  ...
  <user username="tom" password="tom" roles="role1"/>
  <user username="dick" password="dick" roles="role1"/>
  <user username="harry" password="harry" roles="role1"/>
</tomcat-users>
```

2. Define the *HTML FORM for username and password* (`loginform.html`).

```
<html><head><title>Login</title></head>
<body>
<h4>Please login first:</h4>
<form method="post" action="j_security_check">
  Username: <input type="text" name="j_username"/><br/>
  <br/>
  Password: <input type="password" name="j_password"/><br/>
  <br/>
  <input type="submit" value="login"/>
</form>
</body>
</html>
```

Predefined value of `action` *attribute.*

Predefined value of the `name` *attribute in two text fields which represent the username and the password, respectively.*

No servlet is defined to process the form.

3. Define the *HTML error page* which will be shown in case the login fails (`errorpage.html`).

```
<html><head><title>Authorization Failure</title></head>
<body>
<h4>Sorry. Could not log in.<br/>
    Contact your administrator.</h4>
</body>
</html>
```

4. Rest of the setup for authentication is specified (in the indicated order) in the deployment descriptor (`web.xml`) of the web application.

- Define the servlets in the web application in the normal way using the `servlet` element in the `web.xml` file:

```
<web-app ...>
    ...
   <servlet>
     <servlet-name>SearchEngineGUI</servlet-name>
     <servlet-class>SearchEngineGUI</servlet-class>
   </servlet>

   <servlet>
     <servlet-name>SearchHandler</servlet-name>
     <servlet-class>SearchHandler</servlet-class>
   </servlet>
    ...
</web-app>
```

- Define the *security constraint* for the secured servlet in the deployment descriptor (`web.xml`) of the web application:

```
<web-app ...>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>DeclarativeSecurityTestI</web-resource-name>
      <url-pattern>/servlet/SearchEngineGUI</url-pattern>
      <http-method>POST</http-method>
      <http-method>GET</http-method>
    </web-resource-collection>

    <auth-constraint>
      <role-name>role1</role-name>
    </auth-constraint>

    <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  ...
</web-app>
```

A collection of resources to which the security constraint apply.

Only users who have the specified role can access the resource.

Specifies the integrity and confidentiality of data transmission.

- Each `web-resource-collection` element specifies a collection of resources to which the security constraint applies.
  - The `url-pattern` element specifies the URL pattern through which the resource will be accessed.
  - The `http-method` elements specify the HTTP methods that the security constraint will apply to.
- The `auth-constraint` element specifies the roles that can access the resources.
  - The roles specified must be a subset of the roles specified in the `security-role` element (see below).
  - Only users who have this role specified in the server user list can access the resources.
- The `user-data-constraint` element specifies any specific integrity and confidentiality guarantees of the data transmitted (`NONE`, `INTEGRAL`, `CONFIDENTIAL`).

- Specification of the *authentication mechanism* in the `login-config` element.

```
<web-app ...>
     ...
     <login-config>
       <auth-method>FORM</auth-method>
       <form-login-config>
          <form-login-page>/loginform.html</form-login-page>
          <form-error-page>/errorpage.html</form-error-page>
       </form-login-config>
     </login-config>
     ...
</web-app>
```

Method used for authentication

The form for logging in.

The error page to show if login fails.

  – The `auth-method` element specifies the authentication method which can be `BASIC`, `DIGEST`, `CLIENT-CERT`, or `FORM`.
    - If the `BASIC` authentication method is specified, the browser uses predefined GUI dialog box for login and a predefined error page to report any problems with logging in.

- Each role that is legal for the users of this application is specified in a `security-role` element.

```
<web-app ...>
     ...
     <security-role>
       <role-name>role1</role-name>
     </security-role>
     ...
</web-app>
```

Role for users of this application

  – A role specified in a `auth-constraint` element must be one specified in a `security-role` element

Creating WARs: Web Application Deployment

- Using a WAR (Web ARchive) file simplifies web application deployment from development environment to deployment environment.
- All the resources which comprise the application can be bundled in a WAR file.
- A WAR file is a JAR (Java ARchive) file, but it has the extension `.war` instead of the `.jar` extension, and created using the `jar` utility.
- A server treats a WAR file in a special way when such a file is placed in the `webapps` directory.
  - When the server starts up, the WAR file is automatically unpacked by the server and its contents installed in a directory with the same filename as the WAR file without the extension.
- Deploying a WAR file for a web application is a two-step process:
  - Create a WAR file for the file structure under the document root of the web application (here called `myWebApp`):

```
tomcat-home/webapps/myWebApp>jar -cvf myWebApp.war *
```

  - Ship the `myWebApp.war` file, which can be placed under the `webapps` directory to deploy the web application.