CORSO DI LAUREA IN INFORMATICA

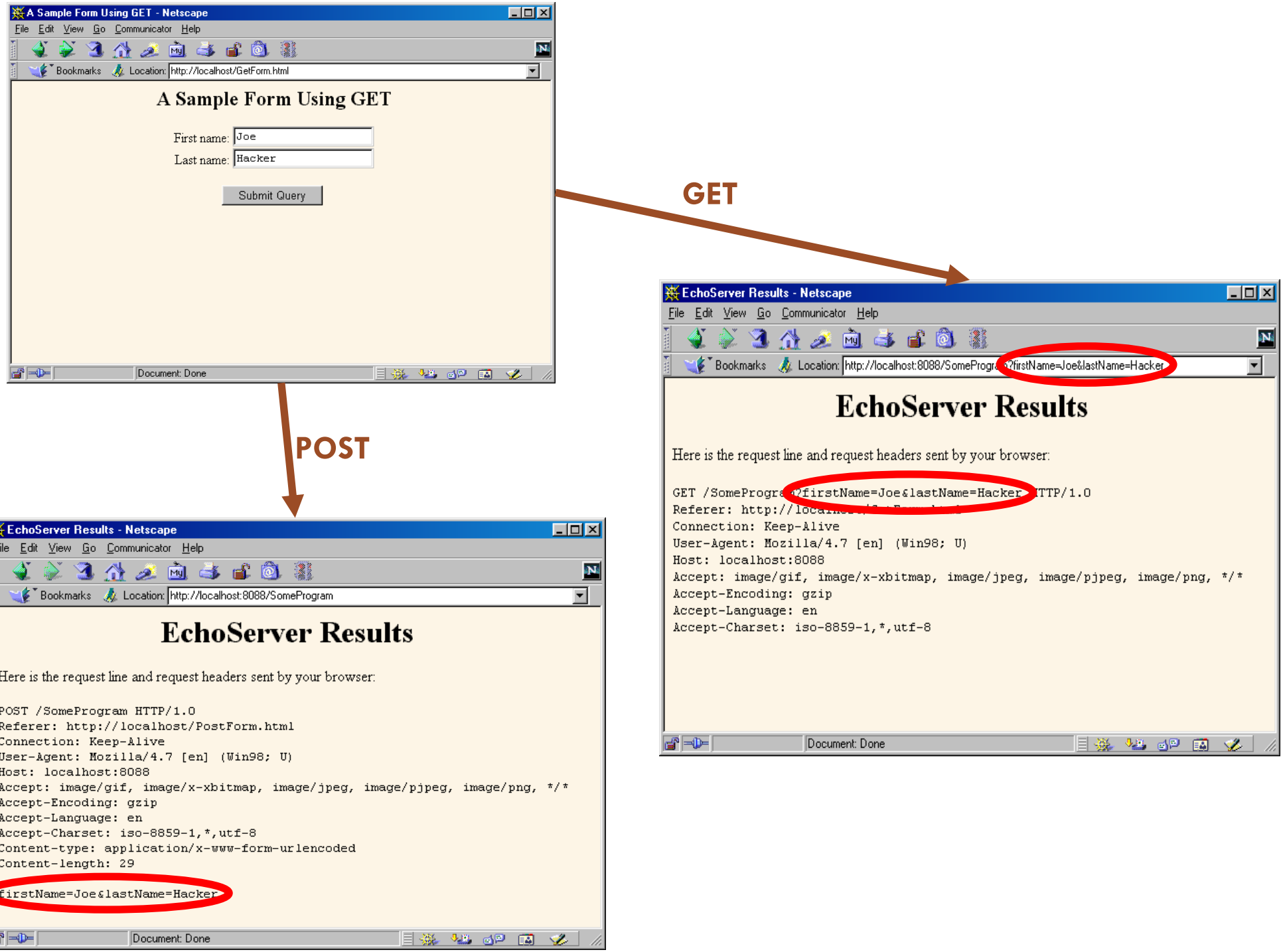
Tecnologie Software per il Web

FORM DATA

a.a. 2019-2020

HTML Forms (getservlet)

```
<!DOCTYPE HTML>
<HTML>
<HEAD><TITLE>A Sample Form Using GET</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2 ALIGN="CENTER">A Sample Form Using GET</H2>

<FORM ACTION="http://localhost:8088/SomeProgram" METHOD="GET">
  First name:
  <INPUT TYPE="TEXT" NAME="firstName" VALUE="Joe"><BR>
  Last name:
  <INPUT TYPE="TEXT" NAME="lastName" VALUE="Hacker"><P>
  <INPUT TYPE="SUBMIT"> <!-- Press this to submit form -->
</FORM>
</BODY></HTML>
```


A Sample Form Using GET - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost/GetForm.html
A Sample Form Using GET
First name: Joe
Last name: Hacker
Submit Query
Document: Done
GET
POST
EchoServer Results - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost:8088/SomeProgram?firstName=Joe&lastName=Hacker
EchoServer Results
Here is the request line and request headers sent by your browser:
GET /SomeProgram?firstName=Joe&lastName=Hacker HTTP/1.0
Referer: http://localhost/GetForm.html
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (Win98; U)
Host: localhost:8088
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Document: Done
EchoServer Results - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://localhost:8088/SomeProgram
EchoServer Results
Here is the request line and request headers sent by your browser:
POST /SomeProgram HTTP/1.0
Referer: http://localhost/PostForm.html
Connection: Keep-Alive
User-Agent: Mozilla/4.7 [en] (Win98; U)
Host: localhost:8088
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 29
firstName=Joe&lastName=Hacker
Document: Done

Reading Form Data in Servlets

- **request.getParameter("name")**
  - Returns URL-decoded value of first occurrence of name in query string
  - Works identically for GET and POST requests
  - Returns **null** if no such parameter is in query
- **request.getParameterValues("name")**
  - Returns an array of the URL-decoded values of all occurrences of name in query string
  - Returns a one-element array if param not repeated
  - Returns **null** if no such parameter is in query
- **request.getParameterNames()**
  - Returns Enumeration of request params

Missing Data

- *What should the Servlet do when the user fails to supply the necessary information?*
- This question has two answers:
  - use default values
  - redisplay the form (prompting the user for missing values)

Check for Missing Data

- Textfield was not in HTML form at all
  - request.getParameter returns **null**
- Textfield was empty when form was submitted
  - request.getParameter returns an **empty String**
- Example check:

```
String value = request.getParameter("fieldName");
if ((value != null) && (!value.equals(""))) {
  //...
}
```

Check for Missing Data (2)

- Always explicitly handle missing or malformed query data

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String headingFont = request.getParameter("headingFont");
    headingFont = replaceIfMissingOrDefault(headingFont, "");
    // ...
    String name = request.getParameter("name");
    name = replaceIfMissing(name, "Lou Zer");
    String title = request.getParameter("title");
    title = replaceIfMissing(title, "Loser");
    String languages = request.getParameter("languages");
    languages = replaceIfMissing(languages, "<I>None</I>");
    String languageList = makeList(languages);
    String skills = request.getParameter("skills");
    skills = replaceIfMissing(skills, "Not many, obviously.");
    // ...
}
```

Check for Missing Data (3)

```
private String replaceIfMissingOrDefault(String orig, String replacement) {
    if ((orig == null) || (orig.trim().equals("")) || (orig.equals("default"))) {
        return (replacement);
    } else {
        return (orig + ", ");
    }
}

private String replaceIfMissing(String orig, String replacement) {
    if ((orig == null) || (orig.trim().equals(""))) {
        return (replacement);
    } else {
        return (orig);
    }
}

private String makeList(String listItems) {
    StringTokenizer tokenizer = new StringTokenizer(listItems, ", ");
    String list = "<UL>\n";
    while (tokenizer.hasMoreTokens()) {
        list = list + "  <LI>" + tokenizer.nextToken() + "\n";
    }
    list = list + "</UL>";
    return (list);
}
```

Filtering Strings for HTML-Specific Characters

- You cannot safely insert arbitrary strings into Servlet output
  - < and > can cause problems anywhere
  - & and " can cause problems inside of HTML attributes
- You sometimes cannot manually translate
  - The string is derived from a program excerpt or another source where it is already in some standard format
  - The string is derived from HTML form data
- Failing to filter special characters from form data makes you vulnerable to cross-site scripting attack

Filtering Strings for HTML-Specific Characters (2)

```
public static String filter(String input) {
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for (int i = 0; i < input.length(); i++) {
        c = input.charAt(i);
        if (c == '<') {
            filtered.append("&lt;");
        } else if (c == '>') {
            filtered.append("&gt;");
        } else if (c == '"') {
            filtered.append("&quot;");
        } else if (c == '&') {
            filtered.append("&amp;");
        } else {
            filtered.append(c);
        }
    }
    return (filtered.toString());
}
```

Servlet That Fails to Filter (code-form-1.html)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>Fail to Filter</h1>\n");
    out.println(getCodeFragment());
}

private String codeFragment =
        "if (a<b) {\n" +
        "  doThis();\n" +
        "} else {\n" +
        "  doThat();\n" +
        "}\n";

public String getCodeFragment() {
    return (codeFragment);
}
```

Correct filtering code-form2.html

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>Correct Filter</h1>\n");
    out.println(filter(getCodeFragment()));
}
```

Check form (JSP) (Fcheck project)

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Check Form</title>
<style>
div.error {
    color: red;
}

div.message {
    color: green;
}
</style>
</head>
<body>
<%
String name = (String)request.getAttribute("name");
if(name == null) name = "";

String surname = (String)request.getAttribute("surname");
if(surname == null) surname = "";


String error = (String)request.getAttribute("error");
if(error != null) {
%>
<div class="error"><%=error %></div>
<%
  }

String message = (String)request.getAttribute("message");
if(message != null) {
%>
<div class="message"><%=message %></div>
<%
  }
%>

<form name="checkformname" method="POST" action="CheckServlet">
Name: <input type="text" name="name" placeholder="Rita" value="<%=name %>"><br>
Surname: <input type="text" name="surname" placeholder="Francese" value="<%=surname %>"><br>
<br>
<input type="submit">
<input type="reset">
</form>

</body>
</html>
```

1. name
2. surname
3. error
4. message

Check form (Servlet)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    String error = "";

    String name = request.getParameter("name");
    String surname = request.getParameter("surname");

    if(name == null || name.trim().equals("")) {
        error+= "Insert name<br>";
    } else {
        request.setAttribute("name", name);
    }

    if(surname == null || surname.trim().equals("")) {
        error+= "Insert surname<br>";
    } else {
        request.setAttribute("surname", surname);
    }

    if(error != null && !error.equals("")) {
        request.setAttribute("error", error);
    } else {
        String message = "Information complete Name:"+name+" Surname:"+surname;
        request.setAttribute("message", message);
    }

    RequestDispatcher dispatcher = getServletContext()
            .getRequestDispatcher("/checkForm.jsp");
    dispatcher.forward(request, response);
}
```

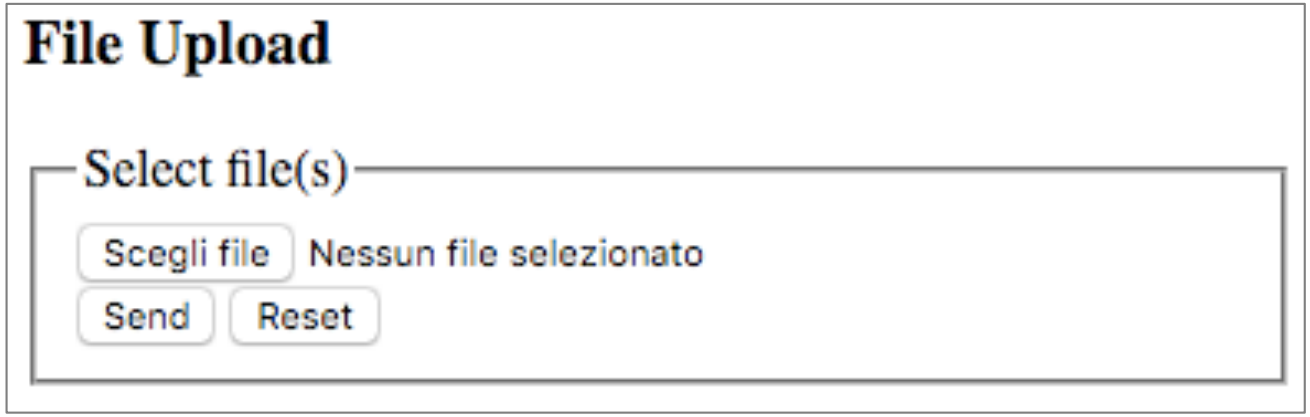
Input file

- Consente di fare l'upload di un file selezionandolo nel file system del client
- Attributi:
  - **type = "file"**
  - **name = text** (specifica il nome del controllo)
  - Richiede una codifica particolare per il form (**multipart/form-data**) perché le informazioni trasmesse con il **post** contengono tipologie di dati diverse: testo per i controlli normali, binario per il file da caricare

```
<form action="http://site.com/bin/adduser" method="post"
   enctype="multipart/form-data" >
   <p>
      <input type="file" name="attach">
   </p>
</form>
```

Form for upload a file

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>File Upload</title>
</head>
<body>
    <h3>File Upload</h3>
    <form method="post" action="fileupload" name="echo" enctype="multipart/form-data">
        <fieldset>
        <legend>Select file(s)</legend>
        <input type="file" name="file" size="50" multiple required /><br>
        <input type="submit" value="Send">
        <input type="reset" value="Reset">
        </fieldset>
    </form>
</body>
</html>
```



Servlet to upload a file

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

@WebServlet(name = "/FileUploadServlet",
            urlPatterns = { "/fileupload" },
            initParams = {@WebInitParam(name = "file-upload", value = "tmpDir") })
@MultipartConfig(fileSizeThreshold = 1024 * 1024 * 2, // 2MB after which the file will be
                                                      // temporarily stored on disk
                 maxFileSize = 1024 * 1024 * 10, // 10MB maximum size allowed for uploaded files
                 maxRequestSize = 1024 * 1024 * 50) // 50MB overall size of all uploaded files
public class FileUploadServlet extends HttpServlet {
    static String SAVE_DIR = "";

    public void init() {
        // Get the file location where it would be stored
        SAVE_DIR = getServletConfig().getInitParameter("file-upload");
    }

    // ...
```

Servlet to upload a file (2)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("text/plain");

    out.write("Error: GET method is used but POST method is required");
    out.close();
}

protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    response.setContentType("text/plain");

    String savePath = request.getServletContext().getRealPath("") + File.separator + SAVE_DIR;

    File fileSaveDir = new File(savePath);
    if (!fileSaveDir.exists()) {
        fileSaveDir.mkdir();
    }

    out.write("upload =\n");
    for (Part part : request.getParts()) {
        String fileName = extractFileName(part);
        if (fileName != null && !fileName.equals("")) {
            part.write(savePath + File.separator + fileName);
            System.out.println(savePath + File.separator + fileName);
            out.write(fileName + "\n");
        }
    }
    out.close();
}
```

Servlet to upload a file (3)

```
private String extractFileName(Part part) {
    //content-disposition: form-data; name="file"; filename="file.txt"
    String contentDisp = part.getHeader("content-disposition");
    String[] items = contentDisp.split(";");
    for (String s : items) {
        if (s.trim().startsWith("filename")) {
            return s.substring(s.indexOf("=") + 2, s.length() - 1);
        }
    }
    return "";
}
```

- **<input type="file" accept="file_extension|audio/*|video/*|image/*">**
- <input type="file" name="photo" multiple accept="image/*">
- <input type="file" name=”document" accept=”.doc,.pdf">
- <input type="file" name=”document" accept=”application/pdf ">

Nel file web.xml

```
<servlet>
   <servlet-name>FileUploadServlet</servlet-name>
   <servlet-class>servlet.FileUploadServlet</servlet-class>
     <init-param>
       <param-name>file-upload</param-name>
       <param-value>tmpDir</param-value>
     </init-param>
   <multipart-config>
      <max-file-size>10485760</max-file-size>
      <max-request-size>20971520</max-request-size>
      <file-size-threshold>5242880</file-size-threshold>
   </multipart-config>
</servlet>
<servlet-mapping>
   <servlet-name>FileUploadServlet</servlet-name>
   <url-pattern>/fileupload</url-pattern>
</servlet-mapping>
```

HTTP Request/Response

- Request

GET /servlet/*SomeName* HTTP/1.1
Host: ...
***Header2*: ...**
...
***HeaderN*: ...**
(Blank Line)

- Response

HTTP/1.1 200 OK
Content-Type: text/html
***Header2*: ...**
...
***HeaderN*: ...**
(Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>

Handling the Client Request: HTTP Request Headers

- Example HTTP 1.1 Request

```
GET /search?keywords=servlets+jsp HTTP/1.1
Accept: image/gif, image/jpg, */*
Accept-Encoding: gzip
Connection: Keep-Alive
Cookie: userID=id456578
Host: www.somebookstore.com
Referer: http://www.somebookstore.com/findbooks.html
User-Agent: Mozilla/4.7 [en] (Win98; U)
```

- **Authorization:** User identification for password-protected pages
- **If-Modified-Since:** Indicates client wants page only if it has been changed after specified date (use getLastModified())

Reading Request Headers (methods in HttpServletRequest)

- General
  - getHeader
  - getHeaders
  - getHeaderNames
- Specialized
  - getCookies
  - getAuthType and getRemoteUser
  - getContentLength
  - getContentType
  - getDateHeader
  - getIntHeader
- Related info
  - getMethod, getRequestURI, getProtocol

Checking for Missing Headers

- HTTP 1.0
  - All request headers are optional
- HTTP 1.1
  - Only Host is required
- Conclusion
  - Always check that request.getHeader is non-null before trying to use it

```
String value = request.getHeader("fieldName");
if (value != null){
  //...
}
```

Printing All Headers

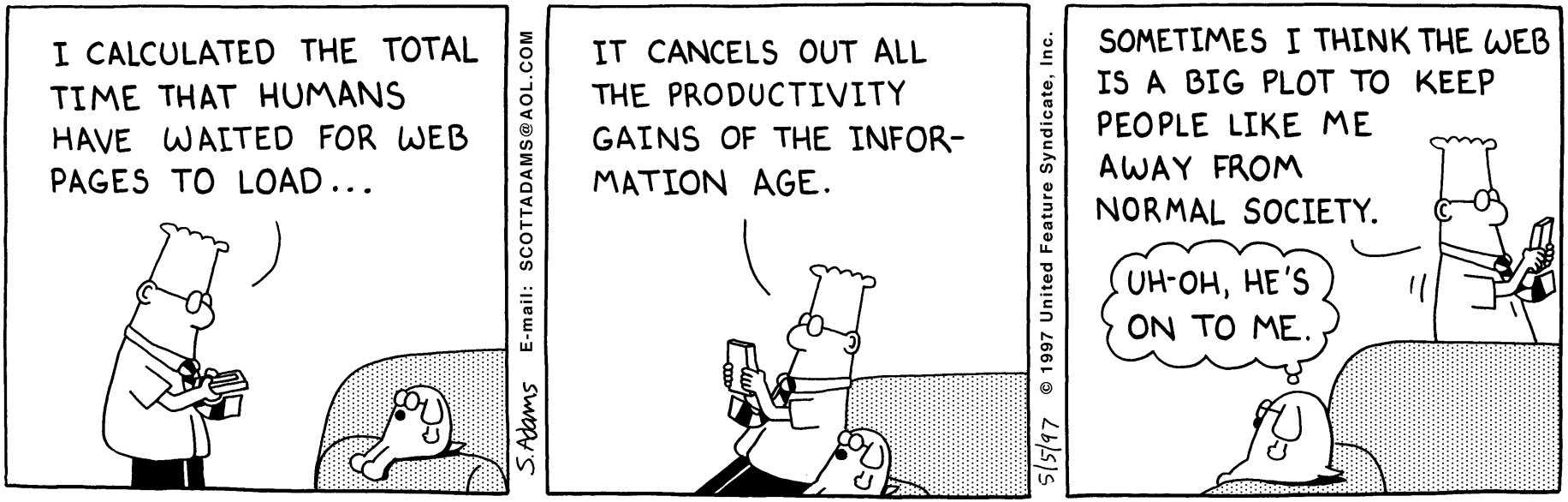
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Servlet Example: Showing Request Headers";
    out.println("<HTML>" +
            "<BODY BGCOLOR=\"#FDF5E6\">" +
            "<H1 ALIGN=CENTER>" + title + "</H1>"
            + "<B>Request Method: </B>" + request.getMethod() + "<BR>"
            + "<B>Request URI: </B>" + request.getRequestURI() + "<BR>"
            + "<B>Request Protocol: </B>" + request.getProtocol() + "<BR><BR>"
            + "<TABLE BORDER=1 ALIGN=CENTER>"
            + "<TR BGCOLOR=\"#FFAD00\">"
            + "<TH>Header Name</TH><TH>Header Value</TH>");
    Enumeration<String> headerNames = request.getHeaderNames();
    while (headerNames.hasMoreElements()) {
        String headerName = (String) headerNames.nextElement();
        out.println("<TR><TD>" + headerName + "</TD><TD>" + request.getHeader(headerName) + "</TD>");
    }
    out.println("</TABLE></BODY></HTML>");
}
```

Servlet Example: Showing Request Headers

Request Method: GET
Request URI: /cercabirraIII/ShowRequestHeaders
Request Protocol: HTTP/1.1

Header Name	Header Value
host	localhost:8080
accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
upgrade-insecure-requests	1
cookie	JSESSIONID=F592168573669DAEFA98C8C7885BAF71
user-agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/603.1.30 (KHTML, like Gecko) Version/10.1 Safari/603.1.30
accept-language	en-us
accept-encoding	gzip, deflate
connection	keep-alive

Sending Compressed Web Pages



- **Gzip** is a text compression scheme that can dramatically reduce the size of HTML (or plain text) pages

Sending Compressed Web Pages (2)

- Most recent browsers know how to handle **gzipped** content
  - so the server can compress the document and send the smaller document over the network
  - after which the browser will automatically reverse the compression (no user action required) and treat the result in the normal manner
- Browsers that support this feature indicate that they do so by setting the **Accept-Encoding** request header

Sending Compressed Web Pages (3)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html");
    String encodings = request.getHeader("Accept-Encoding");
    String encodeFlag = request.getParameter("encoding");
    PrintWriter out;
    String title;
    if ((encodings != null) && (encodings.indexOf("gzip") != -1) && !"none".equals(encodeFlag)) {
        title = "Page Encoded with GZip";
        OutputStream out1 = response.getOutputStream();
        out = new PrintWriter(new GZIPOutputStream(out1), false);
        response.setHeader("Content-Encoding", "gzip");
    } else {
        title = "Unencoded Page";
        out = response.getWriter();
    }

    out.println("<html><body>\n" + "<h1>" + title + "</h1>");
    String line = "Programmazione Web 2016. ";
    for (int i = 0; i < 1000000; i++) {
        out.println(line);
    }
    out.println("</body></html>");
    out.close();
}
```

Sending Compressed Web Pages (4)

- *Uncompressed* (28.8K modem),
  Netscape 4.7 and Internet Explorer 5.0:
  > 50 seconds
- *Compressed* (28.8K modem),
  Netscape 4.7 and Internet Explorer 5.0:
  < 5 seconds

Page Encoded with GZip

Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.
Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016. Programmazione Web 2016.

Differentiating Among Different Browser Types

- The **User-Agent** header identifies the specific browser that is making the request

```
String userAgent = request.getHeader("User-Agent");
if(userAgent != null && userAgent.indexOf("MSIE") != -1) {
    out.print("Microsoft Explorer");
} else {
    out.print("Netscape");
}
```

Substring	Browser
Chrome	Chrome
Apple	Safari
Firefox	Firefox
Netscape	Netscape
MSIE	Explorer
Gecko	Mozilla

HTTP Request/Response

- Request

GET /servlet/*SomeName* HTTP/1.1
Host: ...
***Header2*: ...**
...
***HeaderN*:**
(Blank Line)

- Response

HTTP/1.1 200 OK
Content-Type: text/html
***Header2*: ...**
...
***HeaderN*: ...**
(Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>

Setting Status Codes (SC)

- **response.setStatus(int statusCode)**
  - Use a constant for the code, not an explicit int
  - Constants are in HttpServletResponse
  - Names derived from standard message, e.g., SC_OK, SC_NOT_FOUND, etc.
- **response.sendError(int code, String message)**
  - Wraps message inside small HTML document
- **response.sendRedirect(String url)**
  - Sets status code to 302
  - Sets Location response header also

More on Status code 302

- **302 (Found)**
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - *Servlets should use **sendRedirect**, not setStatus, when setting this header*

- Browsers automatically follow the reference to the new URL given in the Location response header
- Note that the browser reconnects to the new URL immediately; no intermediate output is displayed. This behavior distinguishes redirects from refreshes where an intermediate page is temporarily displayed
- With redirects, another site, not the servlet itself, generates the results

Redirects

- Redirects are useful for the following tasks:
  - **Computing destinations.** If you need to look at the data before deciding where to obtain the necessary results, a redirection is useful
  - **Tracking user behavior.** If you send users a page that contains a hypertext link to another site, you have no way to know if they actually click on the link. But perhaps this information is important in analyzing the usefulness of the different links you send them. So, instead of sending users the direct link, you can send them a link to your own site, where you can then record some information and then redirect them to the real site. For example, several search engines use this trick to determine which of the results they display are most popular

Redirect (2) progetto status-codes

- This servlet sends Internet Explorer users to the Netscape home page, and all other users to the Microsoft home page. The servlet accomplishes this task by using the **sendRedirect** method to send a 302 status code and a Location response header to the browser

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String userAgent = request.getHeader("User-Agent");
    if (userAgent != null && userAgent.contains("MSIE")) {
        response.sendRedirect("http://home.mozilla.com");
    } else {
        response.sendRedirect("http://www.microsoft.com");
    }

}
```

HTTP Request/Response

- Request

GET /servlet/*SomeName* HTTP/1.1
Host: ...
***Header2*: ...**
...
***HeaderN*:**
(Blank Line)

- Response

HTTP/1.1 200 OK
Content-Type: text/html
***Header2*: ...**
...
***HeaderN*: ...**
(Blank Line)
<!DOCTYPE ...>
<HTML>
<HEAD>...</HEAD>
<BODY>
...
</BODY></HTML>

Setting Common Response Headers

- **setContentType**
  - Sets the Content-Type header
  - Servlets almost always use this
  - See table of common MIME types

Common MIME types

Type	Meaning
application/msword	Microsoft Word document
application/octet-stream	Unrecognized or binary data
application/pdf	Acrobat (.pdf) file
application/postscript	PostScript file
application/vnd.ms-excel	Excel spreadsheet
application/vnd.ms-powerpoint	Powerpoint presentation
application/x-gzip	Gzip archive
application/x-java-archive	JAR file
application/x-java-vm	Java bytecode (.class) file
application/zip	Zip archive
audio/basic	Sound file in .au or .snd format
audio/x-aiff	AIFF sound file
audio/x-wav	Microsoft Windows sound file
audio/midi	MIDI sound file
text/css	HTML cascading style sheet
text/html	HTML document
text/plain	Plain text
text/xml	XML document
image/gif	GIF image
image/jpeg	JPEG image
image/png	PNG image
image/tiff	TIFF image
video/mpeg	MPEG video clip
video/quicktime	QuickTime video clip

Writing ResponseHeaders (methods in HttpServletResponse)

- **response.setHeader(String headerName, String headerValue)**
  - Sets an arbitrary header
- **response.setDateHeader(String name, long millisecs)**
  - Converts milliseconds since 1970 to a date string in GMT format
- **response.setIntHeader(String name, int headerValue)**
  - Prevents need to convert int to String before calling setHeader
- **addHeader, addDateHeader, addIntHeader**
  - Adds new occurrence of header instead of replacing
- **setContentLength**
  - Sets the Content-Length header
  - Used for persistent HTTP connections
- **addCookie**
  - Adds a value to the Set-Cookie header

Building Excel Spreadsheets

```
@WebServlet("/Test.xls")
public class Test extends HttpServlet {

    //...

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("application/vnd.ms-excel");
        PrintWriter out = response.getWriter();
        out.println("\tQ1\tQ2\tQ3\tQ4\tTotal");
        out.println("Apples\t78\t87\t92\t29\t=SOMMA(B2:E2)");
        out.println("Oranges\t77\t86\t93\t30\t=SOMMA(B3:E3)");
        out.close();
    }
```

	A	B	C	D	E	F	G
1		Q1	Q2	Q3	Q4	Total	
2	Apples	78	87	92	29	286	
3	Oranges	77	86	93	30	286	
4							
5							
6							

Requirements for Handling Long-Running Servlets

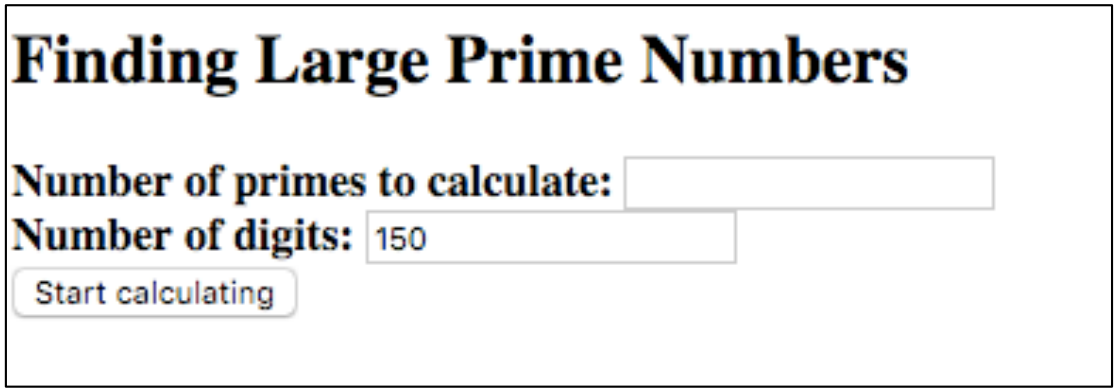
- A way to store data between requests
  - For data that is not specific to any one client, store it in a field (instance variable) of the Servlet
  - For data that is specific to a user, store it in the HttpSession object
  - For data that needs to be available to other Servlets or JSP pages (regardless of user), store it in the ServletContext
- **A way to keep computations running after the response is sent to the user**
  - This task is simple: start a Thread
  - Set the thread priority to a low value so that you do not slow down the server

Persistent Servlet State and Auto-Reloading Pages: Example

- **A way to get the updated results to the browser when they are ready**
  - Unfortunately, because browsers do not maintain an open connection to the server, there is no easy way for the server to proactively send the new results to the browser. Instead, the browser needs to be told to ask for updates. *That is the purpose of the **Refresh** response header*
- **Idea:** generate list of large (e.g., 150-digit) prime numbers
  1. Show partial results until completed
  2. Let new clients make use of results from others
  3. Demonstrates use of the Refresh header
  4. Shows how easy it is for Servlets to maintain state between requests
  5. Also illustrates that Servlets can handle multiple simultaneous connections
     - Each request is in a separate thread

Prime form

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Finding Large Prime Numbers</title>
</head>
<body>
<h2>Finding Large Prime Numbers</h2>
<form action="PrimeNumberServlet" method="get">
    <b>Number of primes to calculate:</b>
    <input type="number" name="numPrimes" value="25" min="1"><br>
    <b>Number of digits:</b>
    <input type="number" name="numDigits" value="150" min="1"><br>
    <input type="submit" value="Start calculating">
</form>

</body>
</html>
```



PrimeNumberServlet

```
@WebServlet("/PrimeNumberServlet")
public class PrimeNumberServlet extends HttpServlet {

    private ArrayList<PrimeList> primeListCollection = new ArrayList<PrimeList>();
    private int maxPrimeLists = 30;

    public static int getIntParameter(HttpServletRequest request, String paramName, int defaultValue) {
        String paramString = request.getParameter(paramName);
        int paramValue;
        try {
            paramValue = Integer.parseInt(paramString);
        } catch (NumberFormatException nfe) { // null or bad format
            paramValue = defaultValue;
        }
        return (paramValue);
    }
```

PrimeNumberServlet (2)

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    int numPrimes = getIntParameter(request, "numPrimes", 50);
    int numDigits = getIntParameter(request, "numDigits", 120);
    PrimeList primeList = findPrimeList(primeListCollection, numPrimes, numDigits);

    if (primeList == null) {
        primeList = new PrimeList(numPrimes, numDigits, true);
        synchronized (primeListCollection) {
            if (primeListCollection.size() >= maxPrimeLists)
                primeListCollection.remove(0);
            primeListCollection.add(primeList);
        }
    }
    List<BigInteger> currentPrimes = primeList.getPrimes();
    int numCurrentPrimes = currentPrimes.size();
    int numPrimesRemaining = (numPrimes - numCurrentPrimes);
    boolean isLastResult = (numPrimesRemaining == 0);
    if (!isLastResult) {
        response.setIntHeader("Refresh", 1);
    }

    ...
```

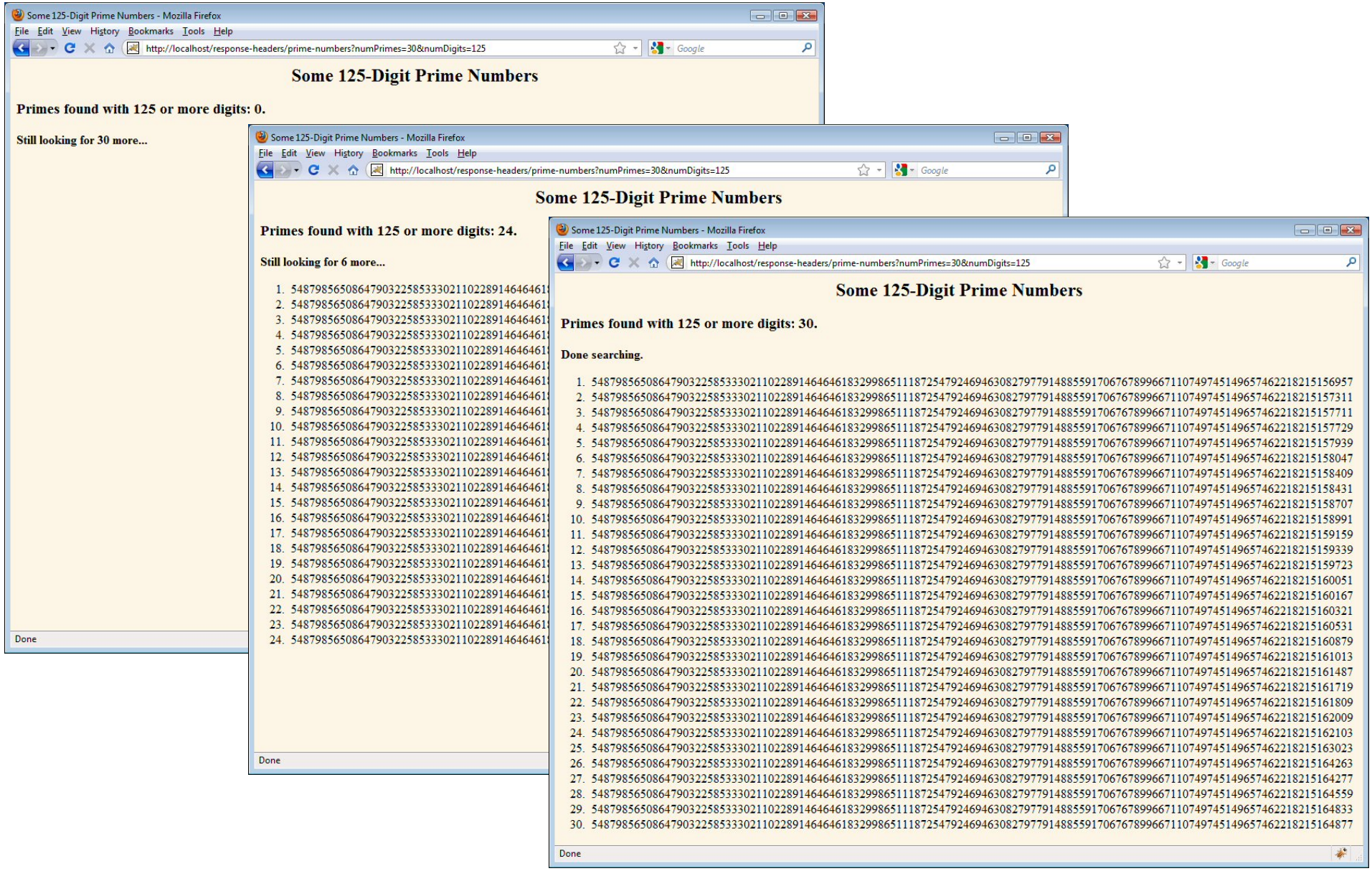
Instantiate a thread that computes the list of primes

PrimeNumberServlet (3)

```
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Some " + numDigits + "-Digit Prime Numbers";
    out.println("<html><body>" +
                "<h2>" + title + "</H2>" +
                "<h3>Primes found with " + numDigits + " or more digits: " + numCurrentPrimes + "</h3>");
    if (isLastResult)
        out.println("<b>Done searching</b>");
    else
        out.println("<b>Still looking for " + numPrimesRemaining + " more...</b>");

    out.println("<ol>");
    for (int i = 0; i < numCurrentPrimes; i++) {
        out.println("<li>" + currentPrimes.get(i) + "</li>");
    }
    out.println("</ol>");
    out.println("</body></htmlL>");

}
```

```
private PrimeList findPrimeList(ArrayList<PrimeList> primeListCollection,
        int numPrimes, int numDigits) {

    synchronized (primeListCollection) {
        for (int i = 0; i < primeListCollection.size(); i++) {
            PrimeList primes = (PrimeList) primeListCollection.get(i);
            if ((numPrimes == primes.numPrimes()) && (numDigits == primes.numDigits()))
                return (primes);
        }
        return (null);
    }
}
```

PrimeNumberServlet (4)



Using Servlets to Generate JPEG Images

- Let us summarize the two main steps Servlets have to perform to build multi-media content
  1. Inform the browser of the content type they are sending
     - To accomplish this task, Servlets set the **Content-Type response header** by using the setContentType method of HttpServletResponse
  2. Send the output in the appropriate format. This format varies among document types, of course, but in most cases you send binary data, not strings as you do with HTML documents
     - Servlets will usually get the **raw output stream** by using the getOutputStream method, rather than getting a PrintWriter by using getWriter

Using Servlets to Generate JPEG Images (2)

1. Create a BufferedImage **image**
2. Draw into the BufferedImage **image**
   - Use normal AWT or Java 2D drawing methods
3. Set the Content-Type response header

```
response.setContentType("image/jpeg");
```

4. Get an output stream

```
OutputStream out = response.getOutputStream
```

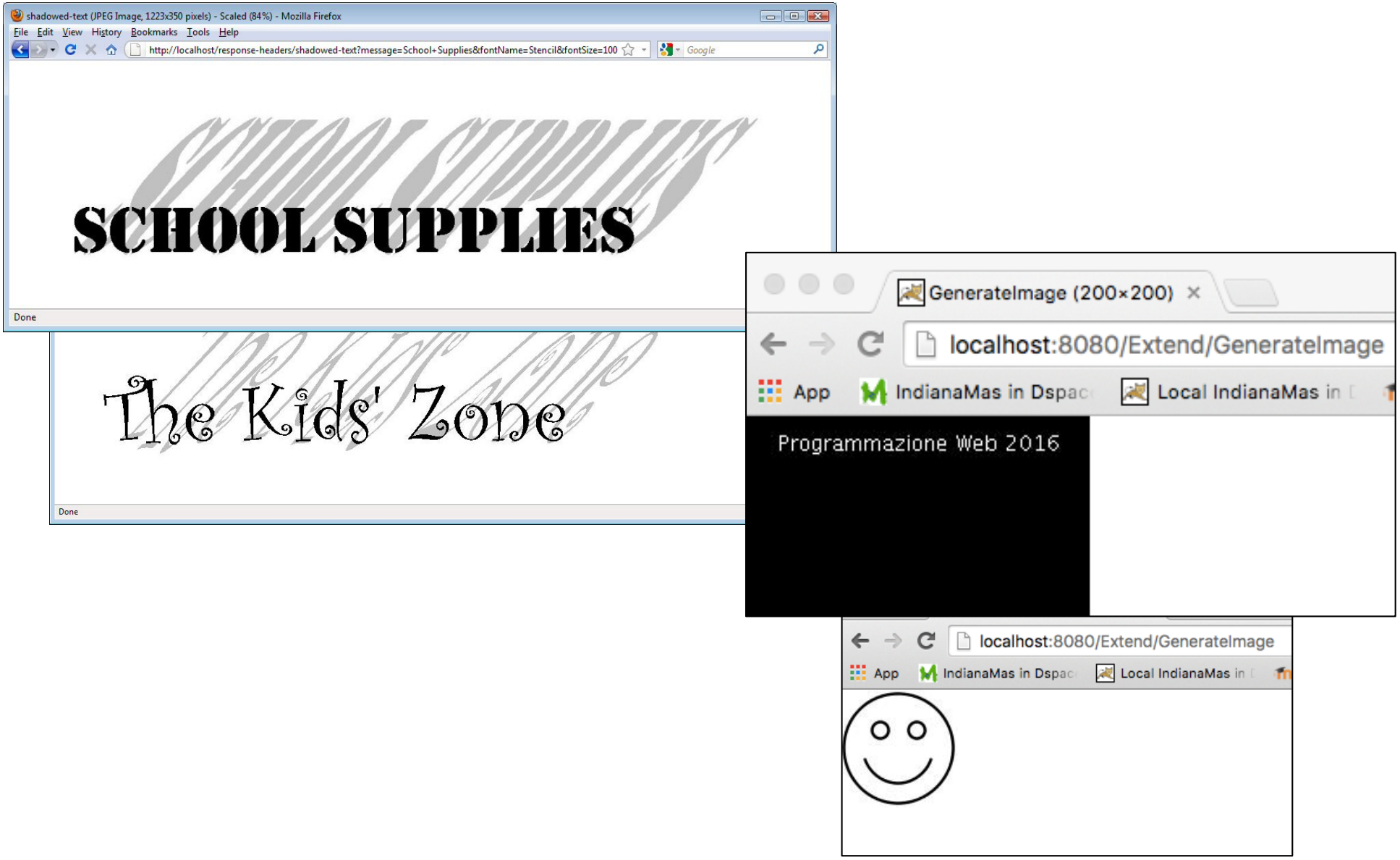
5. Send the BufferedImage in JPEG format to the output stream

```
try {
        ImageIO.write(image, "jpg", out);
} catch(IOException ioe) {
        System.err.println("Error writing JPEG file: ”   + ioe);
}
```

Using Servlets to Generate JPEG Images (3)

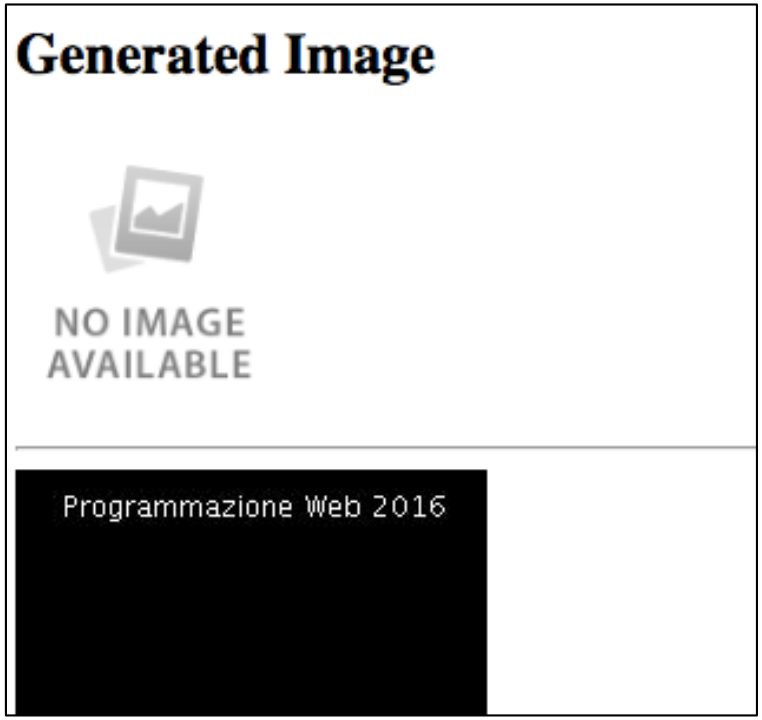
```
@WebServlet("/GenerateImage")
public class GenerateImage extends HttpServlet {

    public static void writeJPEG(BufferedImage image, OutputStream out) {
        try {
            ImageIO.write(image, "jpg", out);
        } catch (IOException ioe) {
            System.err.println("Error outputting JPEG: " + ioe);
        }
    }

    private BufferedImage createImageWithText(){
          BufferedImage bufferedImage = new BufferedImage(200,200,BufferedImage.TYPE_INT_RGB);
          Graphics g = bufferedImage.getGraphics();

          g.drawString("Programmazione Web 2016", 20,20);

          return bufferedImage;
       }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("image/jpeg");
        OutputStream out = response.getOutputStream();
        writeJPEG(createImageWithText(), out);
        out.close();
    }
```

Using Servlets to Generate JPEG Images (3)



Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Generated Image</title>
</head>
<body>
<h2>Generated Image</h2>
    <img src="./BadGenerateImage" onerror="this.src='noImage.png';this.style.width='120px'">
    <hr>
    <img src="./GenerateImage" onerror="this.src='noImage.png'">
</body>
</html>
```



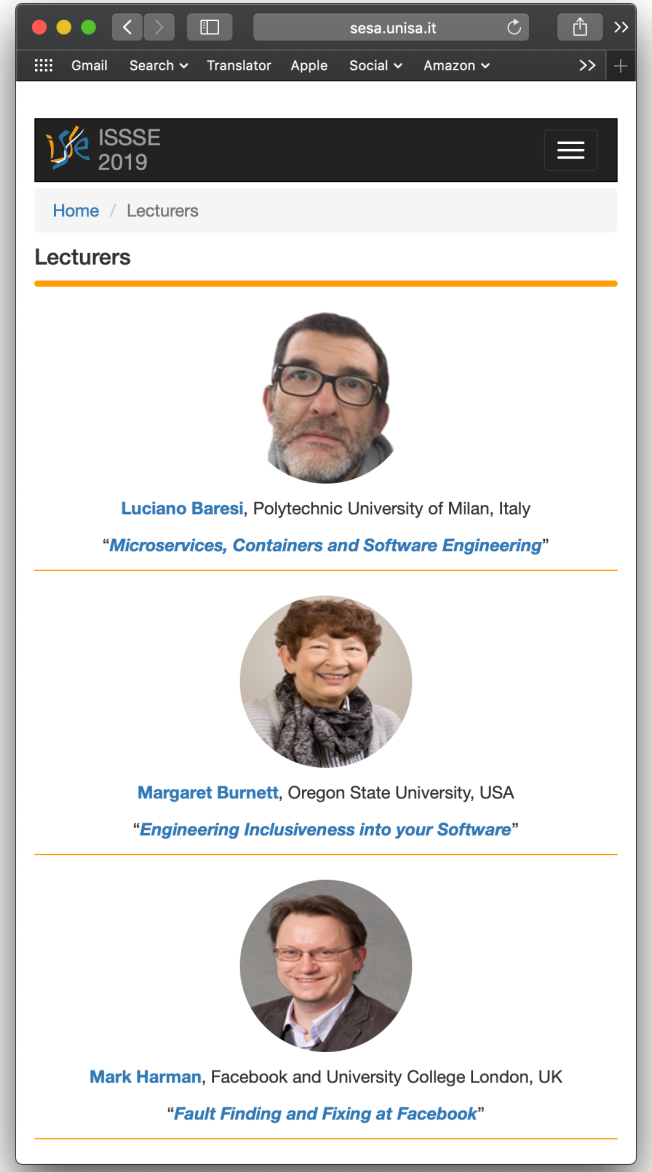
Working with images

DB

```
CREATE TABLE `lectures` (
  `id` int(3) NOT NULL AUTO_INCREMENT,
  /*...*/
  `photo` mediumblob,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

MODEL

```
public class Lecture {

    private int id;

    /* ... */

    public int getId() {
        return this.id;
    }
}
```



VIEW

```
<!-- Lecture mItem -->
<img src="./getPicture?id=<%=mItem.getId() %>" onerror="this.src='./imgs/nophoto.png'">
```

GetPictureServlet (Control)

```
@WebServlet("/getPicture")
public class GetPictureServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        String id = (String) request.getParameter("id");
        if (id != null)
        {
            byte[] bt = PhotoControl.load(id);

            ServletOutputStream out = response.getOutputStream();
            if(bt != null)
            {
                out.write(bt);
                response.setContentType("image /jpeg");
            }
            out.close();

        }
    }
}
```

PhotoControl: load (Utility)

```
public class PhotoControl {
    public synchronized static byte[] load(String id) {

        Connection connection = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;

        byte[] bt = null;

        try {
            connection = DBConnectionPool.getConnection();
            String sql = "SELECT photo FROM lectures WHERE id = " + id;
            stmt = connection.prepareStatement(sql);
            rs = stmt.executeQuery();

            if (rs.next()) {
                bt = rs.getBytes("photo");
            }

        } catch (SQLException sqlException) {
            /* ... */
        }
            finally {
            try {
                if (stmt != null)
                    stmt.close();
            } catch (SQLException sqlException) {
                /* ... */
            } finally {
                if (connection != null)
                    DBConnectionPool.releaseConnection(connection);
            }
        }
        return bt;
    }
}
```

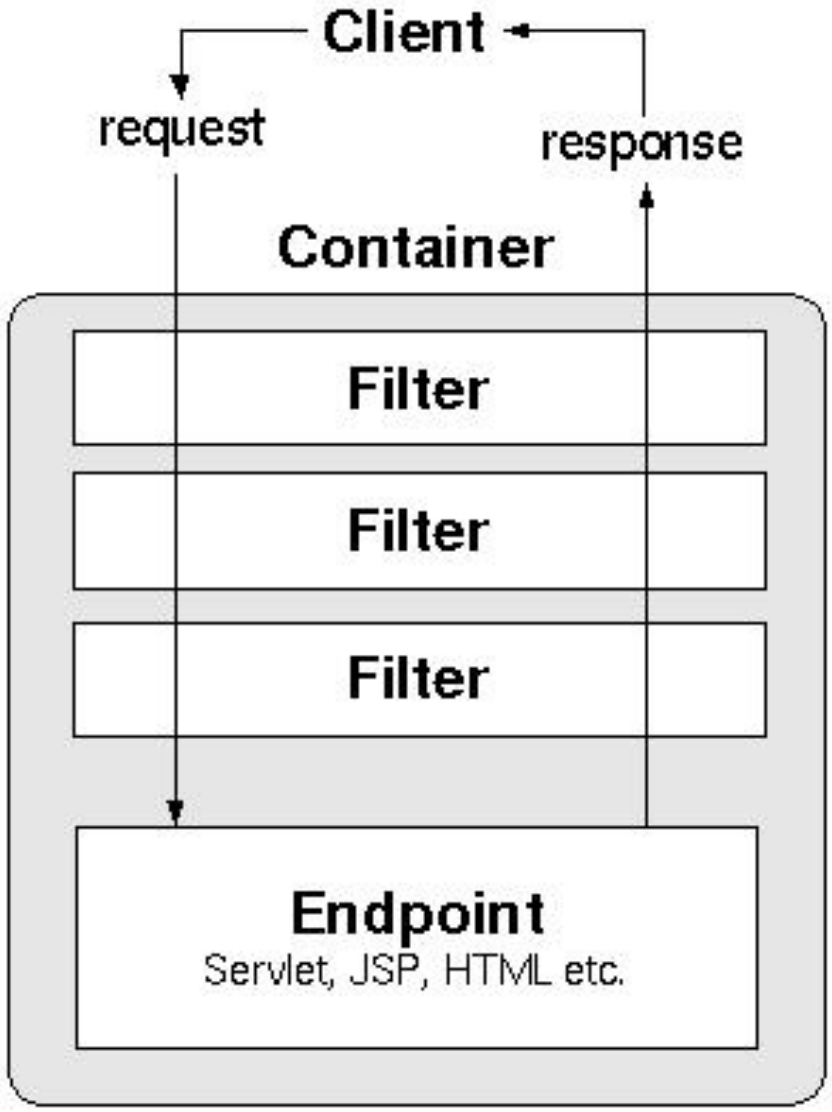
PhotoControl: upload (Utility)

```
public synchronized static void upload(String idA, String photo)
        throws SQLException {
    Connection con = null;
    PreparedStatement stmt = null;

    try {
        con = DBConnectionPool.getConnection();

        stmt = con.prepareStatement("UPDATE lectures SET photo = ? WHERE id = ?");

        File file = new File(photo);
        try {
            FileInputStream fis = new FileInputStream(file);
            stmt.setBinaryStream(1, fis, fis.available());
            stmt.setString(2, idA);

            stmt.executeUpdate();
            con.commit();
        } catch (IOException e) {
            /* ... */
        }
    } finally {
        try {
            if (stmt != null)
                stmt.close();
        } catch (SQLException sqlException) {
            /* ... */
        } finally {
            if (con != null)
                DBConnectionPool.releaseConnection(con);
        }
    }
}
```

1. First, upload the image file on the server
2. Save the image into a specific directory

Servlets: Writing Filters

- **Servlet Filters** are Java classes that can be used in Servlet programming for the following purposes
  - To intercept requests from a client before they access a resource at back end
  - To manipulate responses from server before they are sent back to the client



Types of filters

- There are are various types of filters suggested by the specifications:
  - Authentication Filters
  - Data compression Filters
  - Encryption Filters
  - Filters that Trigger Resource Access Events
  - Input Validation Filters
  - Image Conversion Filters
  - Logging and Auditing Filters
  - MIME-TYPE Chain Filters
  - Tokenizing Filters
  - XSL/T Filters that Transform XML Content

Programming filters

- Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor
  - Alternatively, use the **@WebFilter** annotation to define a filter in a Web application

- When the Web Container starts up your Web application, it creates an instance of each filter that you have declared in the deployment descriptor
  - The filters execute in the order that they are declared in the deployment descriptor

Servlet filter methods

public void doFilter (ServletRequest, ServletResponse, FilterChain)

- This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain

public void init(FilterConfig filterConfig)

- This method is called by the web container to indicate to a filter that it is being placed into service

public void destroy()

- This method is called by the web container to indicate to a filter that it is being taken out of service

Servlet filter: Example

```
public class LogFilter implements Filter {
    public void init(FilterConfig config) throws ServletException {
        // Get init parameter
        String testParam = config.getInitParameter("test-param");

        // Print the init parameter
        System.out.println("Test Param: " + testParam);
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws java.io.IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getServerName();
        out.println("Name " + name + ", Time " + new Date().toString());

        // Pass request back down the filter chain
        chain.doFilter(request, response);
    }

    public void destroy() {
        // Called before the Filter instance is removed from service by the web container
    }
}
```

Servlet filter mapping in web.xml

```
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>LogFilter</filter-class>
  <init-param>
    <param-name>test-param</param-name>
    <param-value>Initialization Parameter</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

- The above filter would apply to all the servlets because we specified /* in the configuration
- A Servlet path applies a filter to few Servlets only
  - Es. **/Admin/***

@WebFilter(filterName = ”LogFilter”, urlPatterns = {"/*"}, initParams = { @WebInitParam(name = ”test-param", value = ”Initialization Parameter")})

Using multiple filters

- Web application may define several different filters with a specific purpose
  - define two filters **AuthenFilter** and **LogFilter**
- Filters application order
  - The order of filter-mapping elements in web.xml determines the order in which the Web container applies the filter to the Servlet
  - Es: apply LogFilter first and then apply AuthenFilter to any Servlet

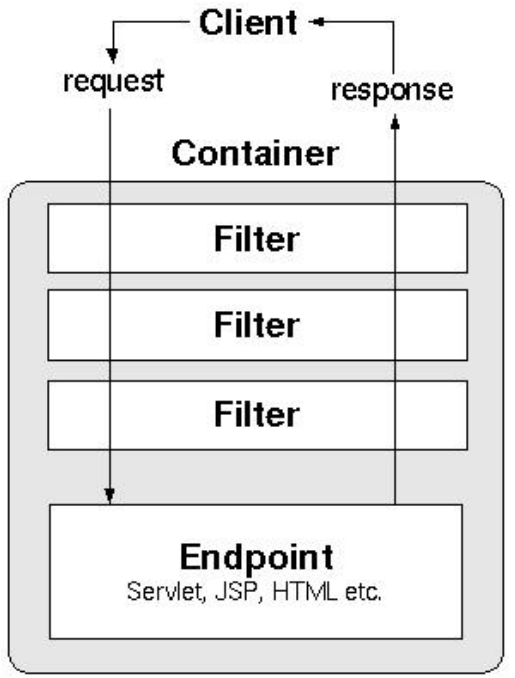
```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
</filter>

<filter>
   <filter-name>AuthenFilter</filter-name>
   <filter-class>AuthenFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>LogFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
   <filter-name>AuthenFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

1

2

Intercepting Request and Response

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {

    doBeforeProcessing(request, response);

    Throwable problem = null;
    try {
        chain.doFilter(request, response);
    } catch (Throwable t) {
        problem = t;
    }

    doAfterProcessing(request, response);

    if (problem != null) {
        if (problem instanceof ServletException) {
            throw (ServletException) problem;
        }
        if (problem instanceof IOException) {
            throw (IOException) problem;
        }
        sendProcessingError(problem, response);
    }
}

private void doBeforeProcessing(ServletRequest request, ServletResponse response) throws IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("Before " + request.getServerName() + ", Time " + new Date().toString());
}

private void doAfterProcessing(ServletRequest request, ServletResponse response) throws IOException {
    PrintWriter out = response.getWriter();
    out.println("After " + request.getServerName() + ", Time " + new Date().toString());
}

private void sendProcessingError(Throwable problem, ServletResponse response) throws IOException {
    //...
}
```



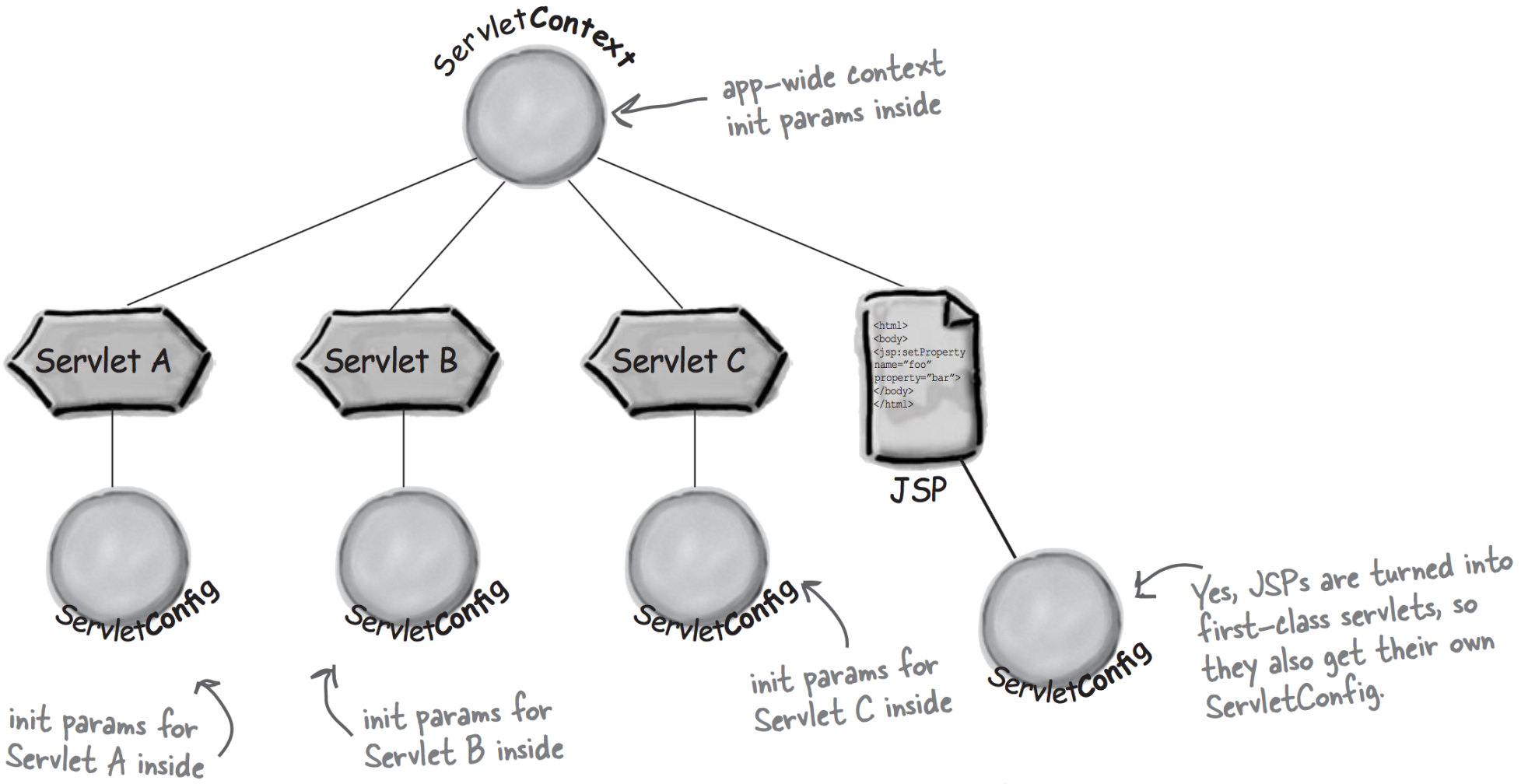
Admin filter (example)

Users:
1. User (login)
2. Admin (login)
3. Guest

```
@WebFilter(urlPatterns = { "/admin/*", "/user/*" })
public class AuthFilter implements Filter {

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
        HttpServletRequest hrequest = (HttpServletRequest) request;
        HttpServletResponse hresponse = (HttpServletResponse) response;

        String uri = hrequest.getRequestURI();

        if (uri.contains("/user/")) {
            HttpSession session = hrequest.getSession(false);

            if (session != null) {
                AccountsModel account = (AccountsModel) session.getAttribute("accountRole");

                if (account != null && (account.isAdmin() || account.isUser())) {
                    chain.doFilter(request, response);
                } else hresponse.sendRedirect(hrequest.getContextPath() + "/login.jsp");
            } else hresponse.sendRedirect(hrequest.getContextPath() + "/login.jsp");
        } else if (uri.contains("/admin/")) {
            HttpSession session = hrequest.getSession(false);

            if (session != null) {
                AccountsModel account = (AccountsModel) session.getAttribute("accountRole");

                if (account != null && account.isAdmin()) {
                    chain.doFilter(request, response);
                } else hresponse.sendRedirect(hrequest.getContextPath() + "/login.jsp");
            } else hresponse.sendRedirect(hrequest.getContextPath() + "/login.jsp");
        } else chain.doFilter(request, response);
    }
}
```

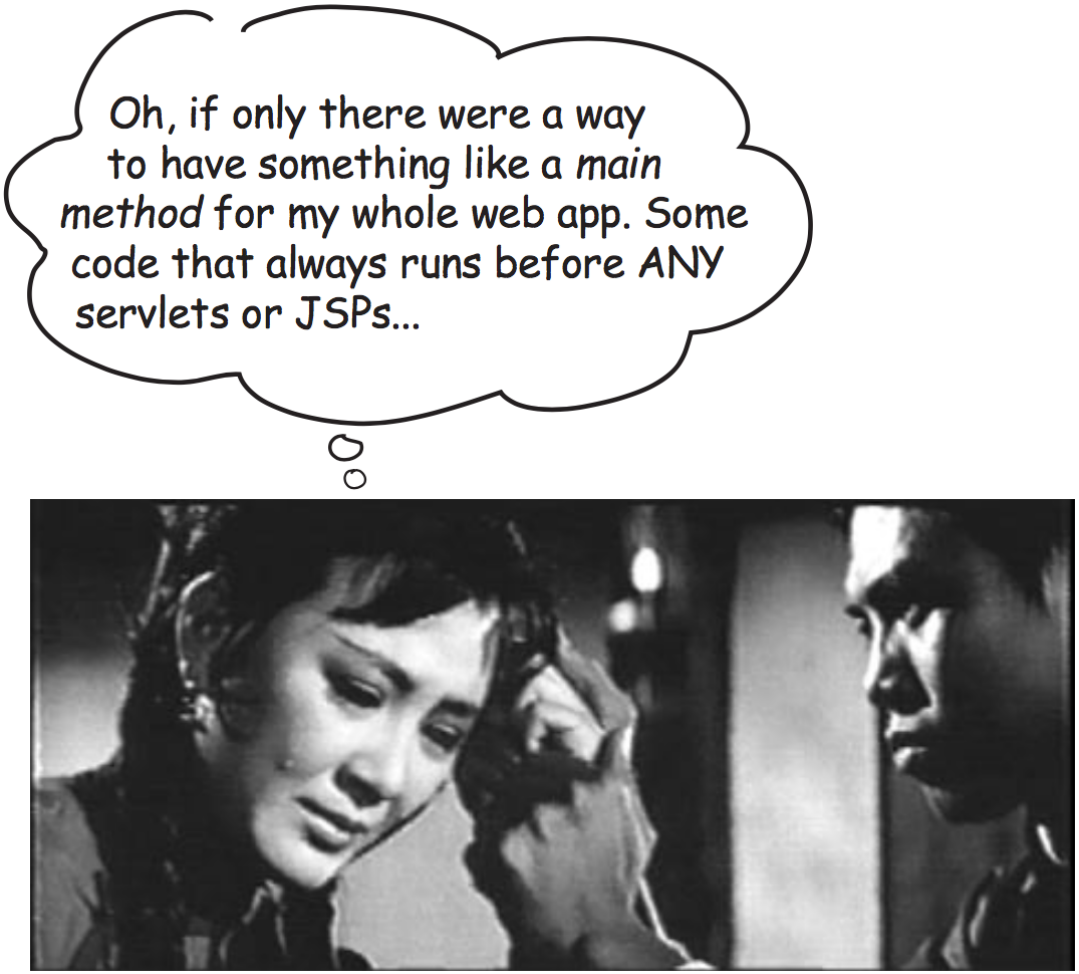
Why this code is not reachable?

ServletConfig is one per servlet
ServletContext is one per web-app



The *main* method…

- She wants to listen for a context initialization event, so that she can get the context init parameters and ***run some code before the rest of the app can service a client***
- She needs something that can be sitting there, waiting to be notified that the app is starting up
- *But which part of the app could do the work? You don't want to pick a servlet—that's not a servlet's job*



The ServletContextListener

- It is a separate class, not a servlet or JSP, that can listen for the two key events in a ServletContext's life - **initialization** (creation) and **destruction**
- This class implements ***javax.servlet.ServletContextListener***

A ServletContextListener class:

```
import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {

   public void contextInitialized(ServletContextEvent event) {
     //code to initialize the database connection
     //and store it as a context attribute
   }

   public void contextDestroyed(ServletContextEvent event) {
     //code to close the database connection
   }
}
```

ServletContextListener is in javax.servlet package.

A context listener is simple: implement ServletContextListener.

These are the two notifications you get. Both give you a ServletContextEvent.

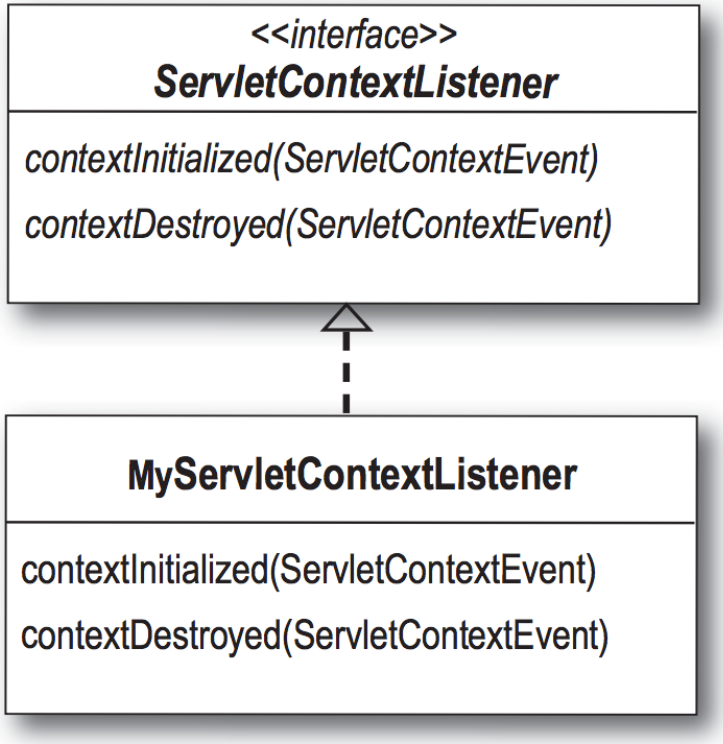
Tutorial

- ***In this example, we'll turn a String init parameter into an actual object—a Dog***
- The listener's job is to get the context init parameter for the dog's breed (Beagle, Poodle, etc.), then use that String to construct a Dog object. The listener then sticks the Dog object into a ServletContext attribute, so that the servlet can retrieve it
    1. The listener object asks the ServletContextEvent object for a reference to the app's ServletContext
    2. The listener uses the reference to the ServletContext to get the context init parameter for "breed", which is a String representing a dog breed
    3. The listener uses that dog breed String to construct a Dog object
    4. The listener uses the reference to the ServletContext to *set* the Dog attribute in the ServletContext
    5. The tester servlet in this web app *gets* the Dog object from the ServletContext, and calls the Dog's getBreed() method

Making and using a context listener

- ***Configure a listener through the web.xml Deployment Descriptor***

① Create a listener class



Put a <listener> element in the web.xml Deployment Descriptor

```
<listener>
  <listener-class>
     com.example.MyServletContextListener
  </listener-class>
</listener>
```

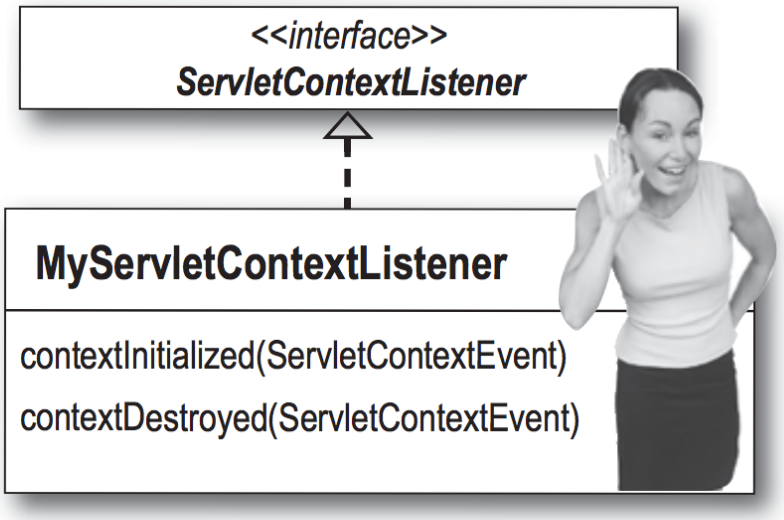
```
@WebListener
public class MyServletContextListener implements ServletContextListener {
  …
}
```

We need three classes and one DD

① The ServletContextListener

MyServletContextListener.java

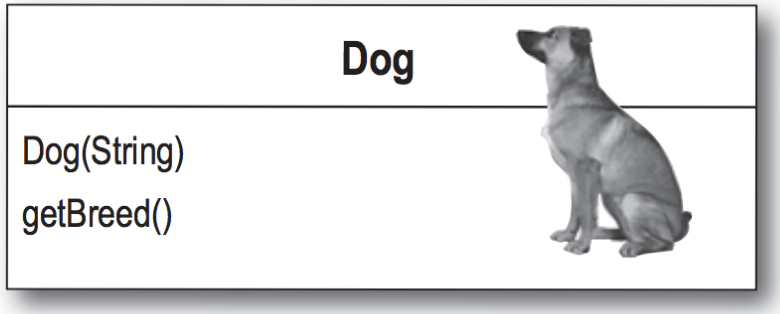
This class implements ServletContextListener, gets the context init parameters, creates the Dog, and sets the Dog as context attribute.



② The attribute class

Dog.java

The Dog class is just a plain old Java class. Its job is to be the attribute value that the ServletContextListener instantiates and sets in the ServletContext, for the servlet to retrieve.



MyServletContextListener and Dog

WEB.XML

```
<context-param>
  <param-name>breed</param-name>
  <param-value>Labrador</param-value>
</context-param>
```

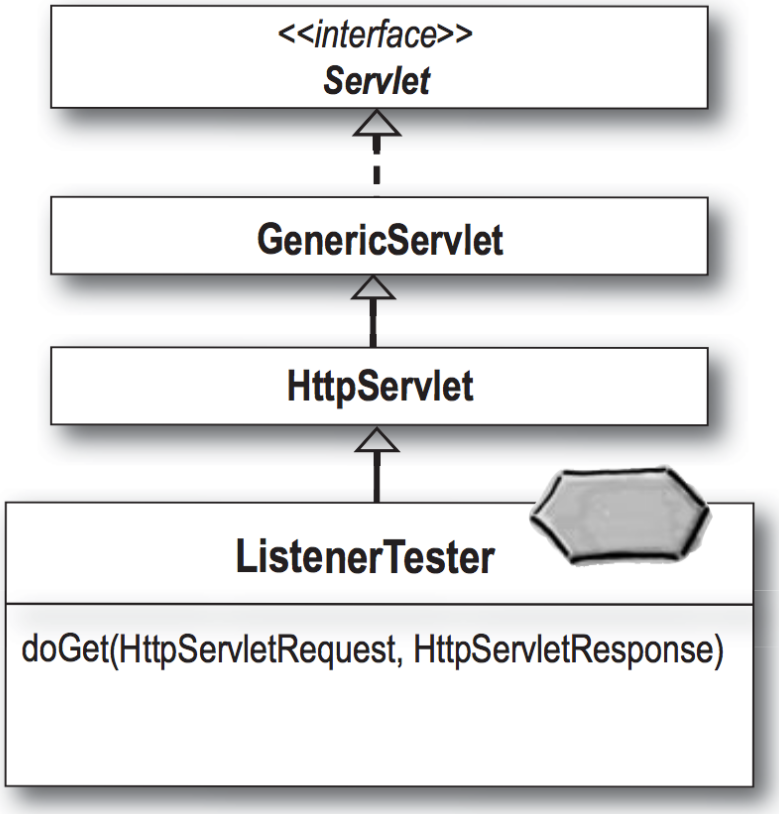
```
public class Dog {
    private String breed;

    public Dog(String breed) {
        this.breed = breed;
    }

    public String getBreed() {
        return breed;
    }
}
```

```
@WebListener
public class MyServletContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent event)  {
        // Initialize database connection
        // Store it as a context attribute
        // Use a Dog object to illustrate:
        ServletContext sc = event.getServletContext();
        String dogBreed = sc.getInitParameter("breed");
        Dog d = new Dog(dogBreed);
        sc.setAttribute("dog", d);

        System.out.println("Initialized: "+event.getServletContext().getServerInfo());
    }

    public void contextDestroyed(ServletContextEvent event)  {
        // Close the database connection
        System.out.println("Destroyed: "+event.getServletContext().getServerInfo());
    }
}
```

③ The Servlet

ListenerTester.java

This class extends HttpServlet. Its job is to verify that the listener worked by getting the Dog attribute from the context, invoking getBreed() on the Dog, and printing the result to the response (so we'll see it in the browser).



ListenerTester

```
@WebServlet("/ListenerTester")
public class ListenerTester extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<br><br>");
        out.println("Test context attributes set by MyServletContextListener<br>");
        out.println("<br>");
        Dog dog = (Dog) getServletContext().getAttribute("dog");

        out.println("Dog's breed is " + dog.getBreed());
        out.println("<br><br>");
    }
}
```

Attributes & listeners

- An attribute is an object set (referred to as *bound*) into one of three other servlet API objects—ServletContext, HttpSession , HttpServletRequest (or ServletRequest). You can think of it as simply a name/value pair (where the name is a String and the value is an Object) in a map instance variable
- Who can get and set the attributes?
- *You can listen for events related to context events, context attributes, servlet requests and attributes, and HTTP sessions and session attributes*

Event Listener Categories

- The event interfaces are as follows:
  1. **ServletRequestListener,** receives notifications for ServletRequest init and destroy.
  2. **ServletRequestAttributeListener,** is used for receiving notification events about ServletRequest attribute changes.
  3. **ServletContextListener,** receives notifications for ServletContext init and destroy.
  4. **ServletContextAttributeListener,** is used for receiving notification events about ServletContext attribute changes.
  5. **HttpSessionListener,** can be used to get notified when a HTTP session is created and destroyed.
  6. **HttpSessionAttributeListener,** can be implemented to get notified when attributes to HttpSession are added or removed.
  7. **HttpSessionBindingListener,** can be used to get notifications when an instance is added to the session or when it is removed from the session.
  8. **HttpSessionActivationListener,** is used for responding to events when a sessions object migrates from one VM to another.