CORSO DI LAUREA IN INFORMATICA

Tecnologie Software per il Web

FORM DATA, HEADER, FILTER E LISTENER

Docente: prof. Romano Simone

a.a. 2024-2025

HTML Forms

```
<!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>
```

POST

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

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

POST

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
```

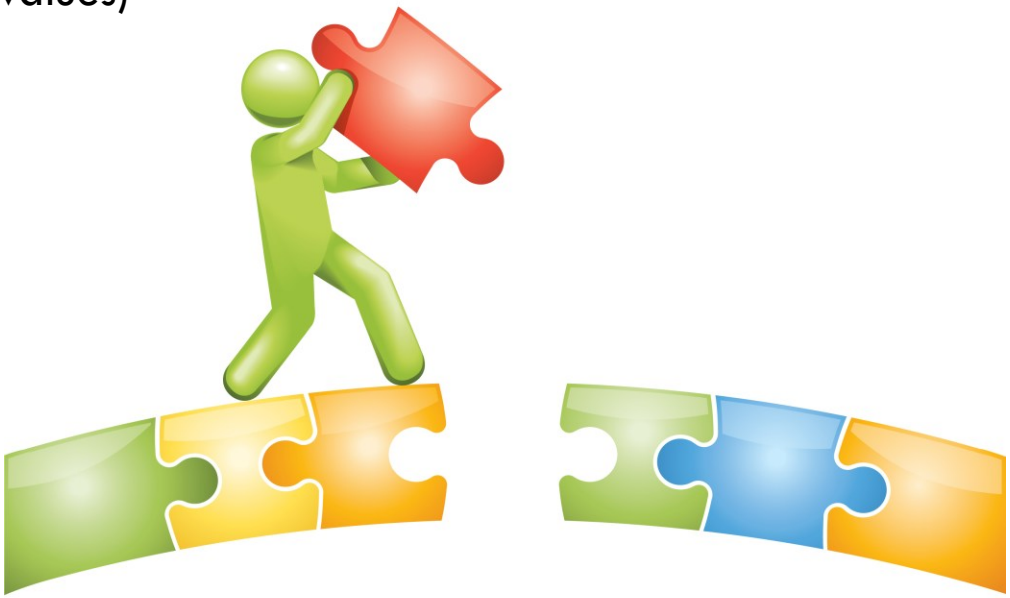
Document: Done

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:
  - In some cases, it is reasonable to use default values
  - In other cases, it is needed to 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 (“”)**
- Textfield contained one or more spaces when the form was submitted
  - request.getParameter returns a **String containing one or more spaces (“ ”)**
- Example check:

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

Check for Missing Data (see CheckServlet.java in FormCheck.zip)

- Always explicitly handle missing or malformed query data

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

    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 {
            name = name.trim();
            request.setAttribute("name", name);
        }

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

        if (!error.equals("")) {
            request.setAttribute("error", error);
        }
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/checkForm.jsp");
        dispatcher.forward(request, response);
    }
```

Check for Missing Data (see checkForm.jsp in FormCheck.zip)

```
<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>
<%  }
    if (name != "" && surname != "") { %>
    Name: <%=name %> Surname: <%=surname %>
<%  }
    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="Your name" value="<%=name %>"><br>
Surname: <input type="text" name="surname" placeholder="Your surname" value="<%=surname %>"><br>
<br>
<input type="submit">
<input type="reset">
</form>
</body>
```

Filtering Strings for HTML-Specific Characters

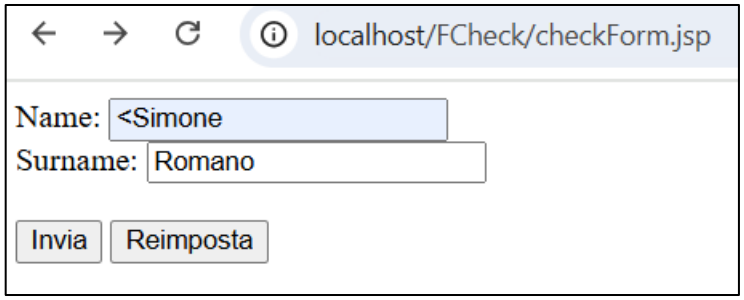
- You cannot safely insert arbitrary strings into Servlet output
  - < and > can cause problems anywhere
  - **&, “, and ‘** can cause problems inside of HTML attributes

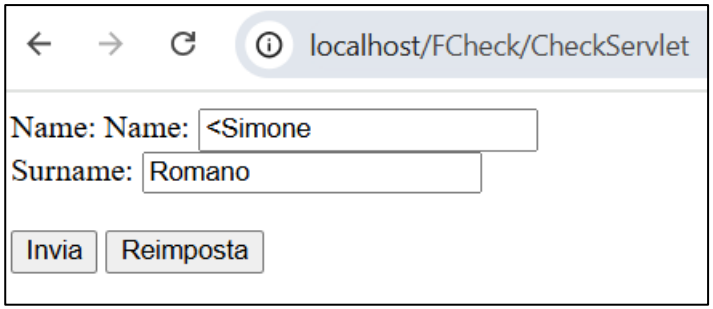
Filtering Strings for HTML-Specific Characters (2)

```
public class HtmlDecoder {

    public static String encodeHtmlEntities(String input) {
        if (input == null) return null;

        return input.replace("&", "&amp;")
                    .replace("<", "&lt;")
                    .replace(">", "&gt;")
                    .replace("\"", "&quot;")
                    .replace("'", "&#39")
                    .replace(" ", " ");
    }
}
```

Servlet That Fails to Filter

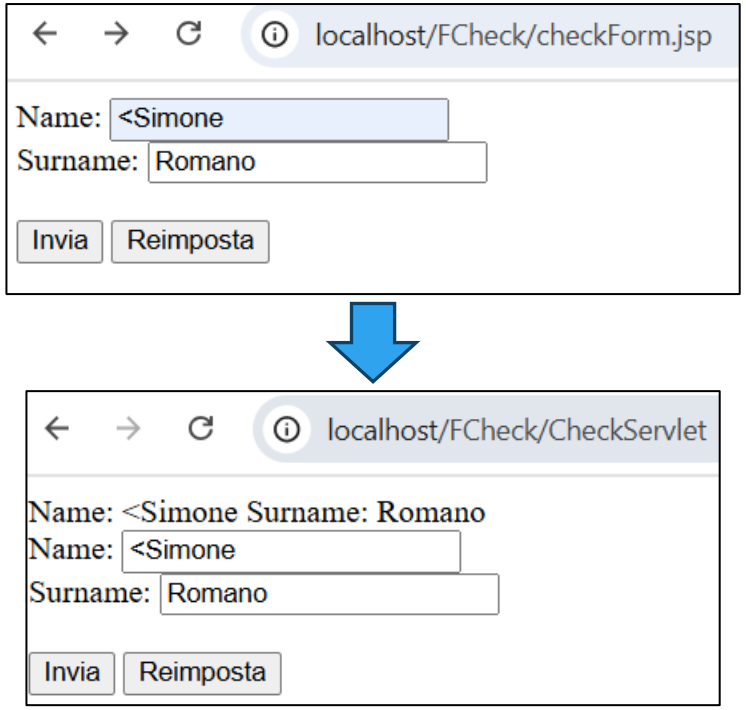
```java
@WebServlet("/CheckServlet")
public class CheckServlet extends HttpServlet {

    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 {
            name = name.trim();
            request.setAttribute("name", name);
        }

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

        if (!error.equals("")) {
            request.setAttribute("error", error);
        }
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/checkForm.jsp");
        dispatcher.forward(request, response);
    }
```





Correct filtering

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

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String error = "";
        String name = request.getParameter("name");
        name = HtmlDecoder.encodeHtmlEntities(name);
        String surname = request.getParameter("surname");
        surname = HtmlDecoder.encodeHtmlEntities(surname);
        if (name == null || name.trim().equals("")) {
            error += "Insert name<br>";
        } else {
            name = name.trim();
            request.setAttribute("name", name);
        }
        if (surname == null || surname.trim().equals("")) {
            error += "Insert surname<br>";
        } else {
            surname = surname.trim();
            request.setAttribute("surname", surname);
        }
        if (!error.equals("")) {
            request.setAttribute("error", error);
        }
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/checkForm.jsp");
        dispatcher.forward(request, response);
    }
```



Input file (FileUpload.zip)

- 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

```
<%
   String message = (String)request.getAttribute("message");
   String error = (String)request.getAttribute("error");
%>
    <h3>JSP File Upload</h3>

<% if(message != null && !message.equals("")) { %>
    <p><%=message %>
<% } %>

    <form method="post" action="fileupload" enctype="multipart/form-data">
        <fieldset>
        <legend>Select file(s)</legend>
        <input type="file" name="file" multiple /><br>
        <input type="submit" value="Send">
        <input type="reset" value="Reset">
        </fieldset>
    </form>
<% if(error != null && !error.equals("")) { %>
    <p style="color:red;"><%=error %>
<% } %>
</body>
</html>
```

JSP File Upload

Select file(s)

Scegli file Nessun file selezionato

Send Reset

Servlet to upload a file

```
import java.io.File;

@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 {

    private static final long serialVersionUID = 1L;
    static String SAVE_DIR = "";

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

    public FileUploadServlet() {
        super();
    }
```

Servlet to upload a file (2)

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String savePath = request.getServletContext().getRealPath("") + SAVE_DIR;
        File fileSaveDir = new File(savePath);
        if (!fileSaveDir.exists()) {
            fileSaveDir.mkdir();
        }
        String message = "upload =\n";
        if (request.getParts() != null) {
            for (Part part : request.getParts()) {
                String fileName = part.getSubmittedFileName();
                if (fileName != null && !fileName.equals("")) {
                    part.write(savePath + File.separator + fileName);
                    System.out.println(savePath + File.separator + fileName);
                    message = message + " " + fileName + "\n";
                } else {
                    request.setAttribute("error", "Errore: Bisogna selezionare almeno un file");
                }
            }
        }
        request.setAttribute("message", message);
        RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/index.jsp");
        dispatcher.forward(request, response);
    }
}
```

Se si usasse il 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>
      <file-size-threshold>2097152</file-size-threshold>
      <max-file-size>10485760</max-file-size>
      <max-request-size>52428800</max-request-size>
   </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>

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){
  //...
}
```

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");
}
```

User-Agent header value when using Microsoft Explorer (up to 11):

```
Mozilla/5.0 (compatible;
MSIE 10.0; Windows NT 6.2;
Trident/6.0)
```

Substring	Browser
Chrome	Chrome
Safari	Safari
Firefox	Firefox
MSIE	Explorer
…	…

Building Excel Spreadsheets (Excel.zip)

```
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)");
}
```

	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							

Setting Common Response Headers

- **setContentType**
  - Sets the Content-Type header
  - 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

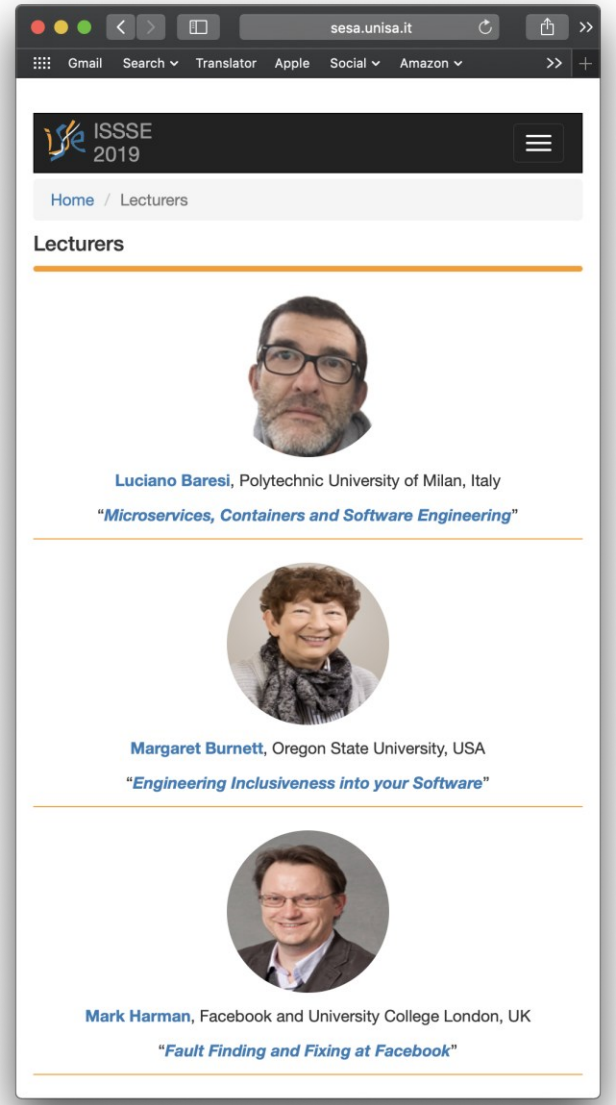
Working with images (PhotoDynamic.zip)

DB

```
CREATE TABLE lectures (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL DEFAULT '',
  `surname` varchar(100) NOT NULL DEFAULT '',
  `photo` mediumblob DEFAULT NULL,
  PRIMARY KEY (`id`)
)
```

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 {
    private static final long serialVersionUID = 1L;

    public GetPictureServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        String id = (String) request.getParameter("id");
        if (id != null)
        {
            byte[] bt = DAOPhoto.load(Integer.parseInt(id));

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

Retrive image from database

DAOPhoto: load (Utility)

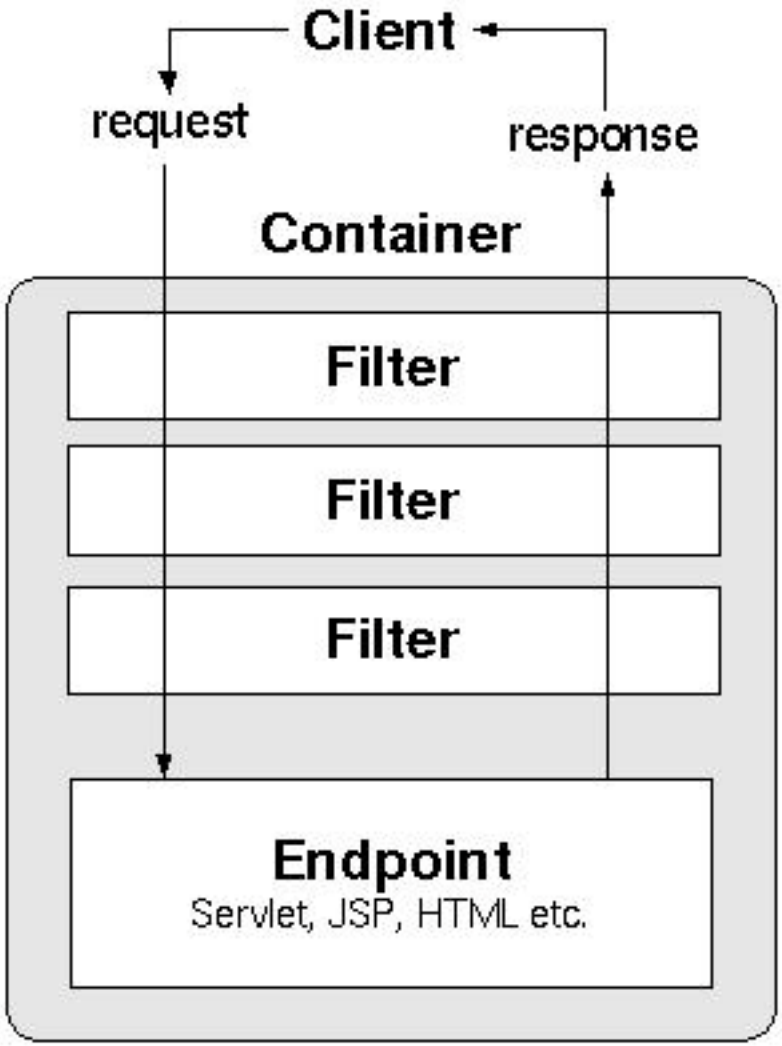
```java
public class DAOPhoto {
    public synchronized static byte[] load(int id) {
        Connection connection = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        byte[] bt = null;
        try {
            connection = DBConnectionPool.getConnection();
            String sql = "SELECT photo FROM lectures WHERE id = ?";
            stmt = connection.prepareStatement(sql);
            stmt.setInt(1, id);
            rs = stmt.executeQuery();
            if (rs.next()) {
                bt = rs.getBytes("photo");
            }
        } catch (SQLException sqlException) {
            System.out.println(sqlException);
        } finally {
            try {
                if (stmt != null)
                    stmt.close();
            } catch (SQLException sqlException) {
                System.out.println(sqlException);
            } finally {
                if (connection != null)
                    DBConnectionPool.releaseConnection(connection);
            }
        }
        return bt;
    }
```

DAOPhoto: upload (Utility)

```
public synchronized static void updatePhoto(int id, InputStream photo) throws SQLException {
    Connection con = null;
    PreparedStatement stmt = null;
    try {
        con = DBConnectionPool.getConnection();
        stmt = con.prepareStatement("UPDATE lectures SET photo = ? WHERE id = ?");
        try {
            stmt.setBinaryStream(1, photo, photo.available());
            stmt.setInt(2, id);
            stmt.executeUpdate();
        } catch (IOException e) {
            System.out.println(e);
        }
    } finally {
        try {
            if (stmt != null)
                stmt.close();
        } catch (SQLException sqlException) {
            System.out.println(sqlException);
        } finally {
            if (con != null)
                DBConnectionPool.releaseConnection(con);
        }
    }
}
```

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.zip)

- 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")})

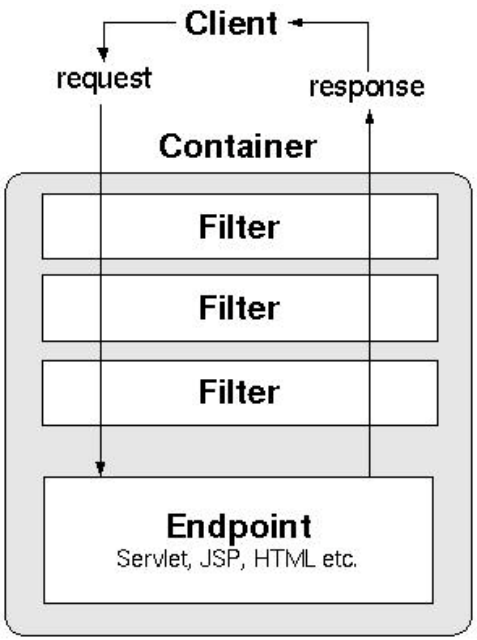
Using multiple filters

- Web application may define several different filters with a specific purpose
  - define two filters **LogFilter** and **AuthenFilter**
- 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

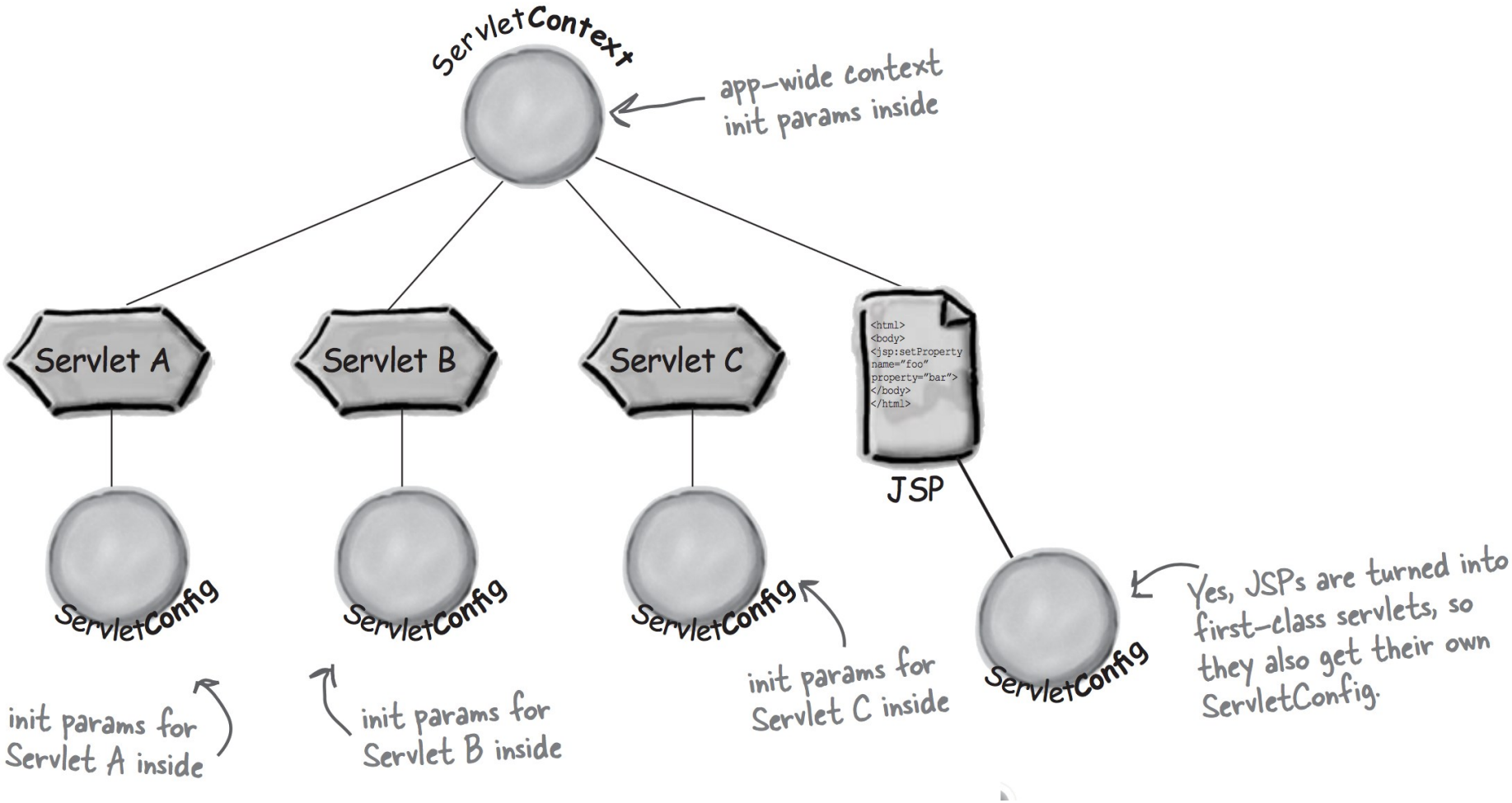
```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>LogFilter</filter-class>
</filter>

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

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

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

Intercepting Request and Response (see LogFilter.java in Filters.zip)

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

Before  doBeforeProcessing(request, response);

------  chain.doFilter(request, response);

After   doAfterProcessing(request, 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());
    }
```

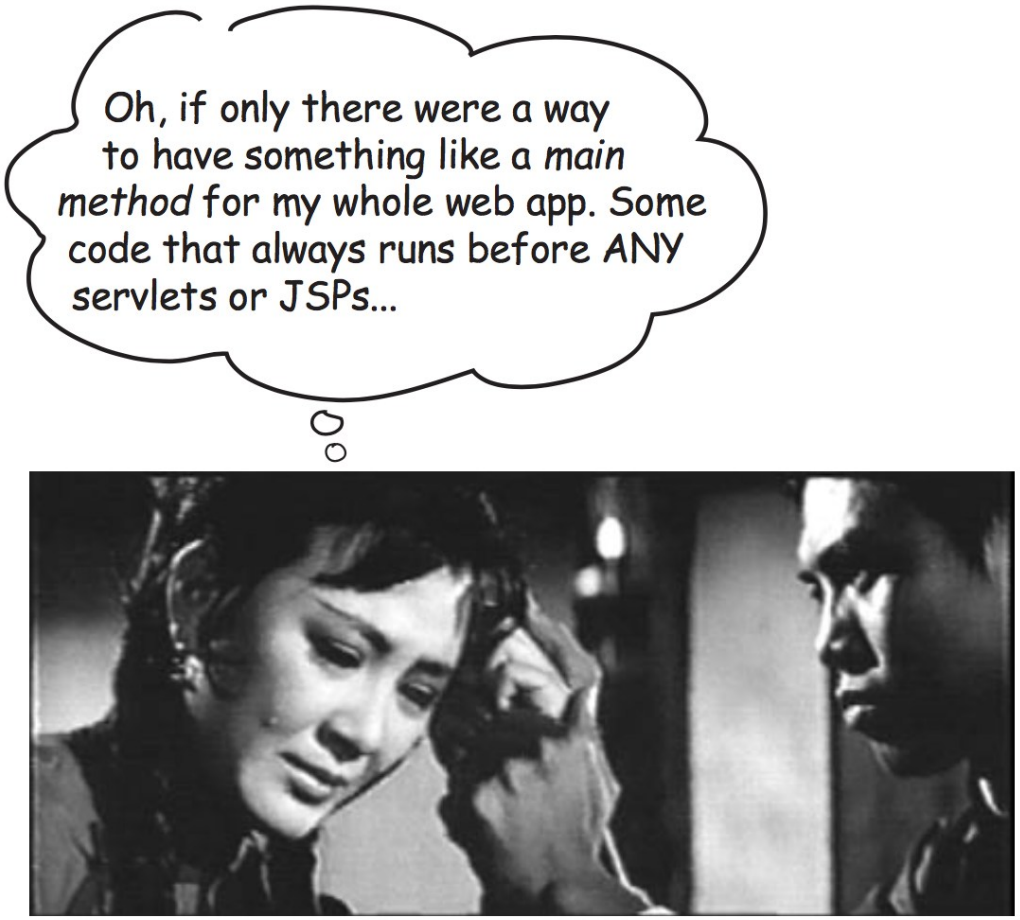


ServletConfig is one per servlet
ServletContext is one per web-app



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 ***jakarta.servlet.ServletContextListener***

A ServletContextListener class:

A context listener is simple: implement ServletContextListener.

```
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
   }
}
```

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

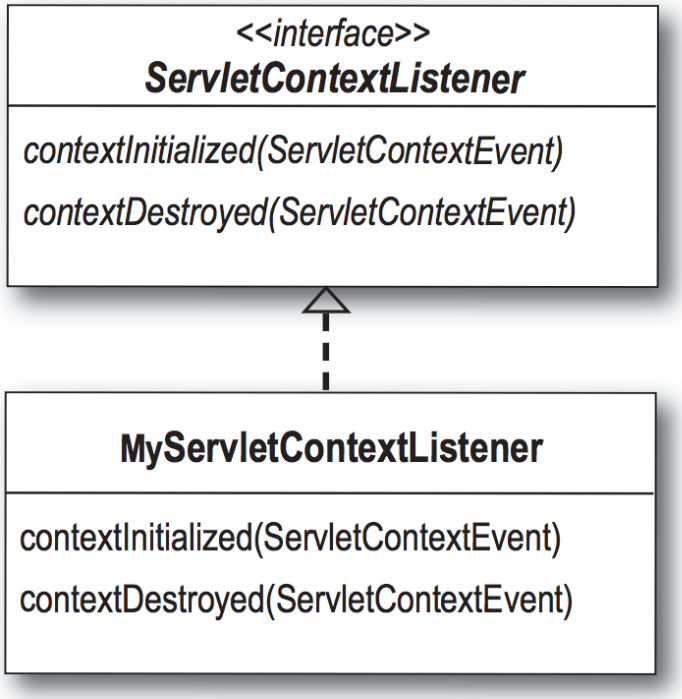
Tutorial

- ***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 (**application**), 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>
```

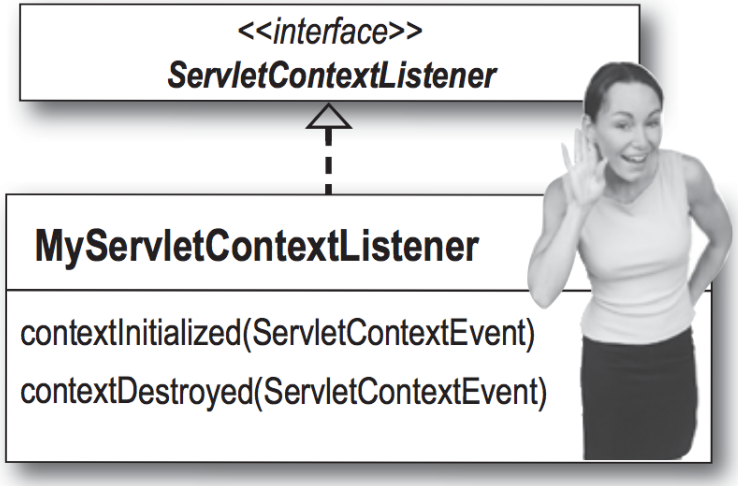
```
@WebListener
public class MyServletContextListener implements ServletContextListener {
  ...
}
```

We need three classes and one DD

① The ServletContextListener

MyServletContextListener.java

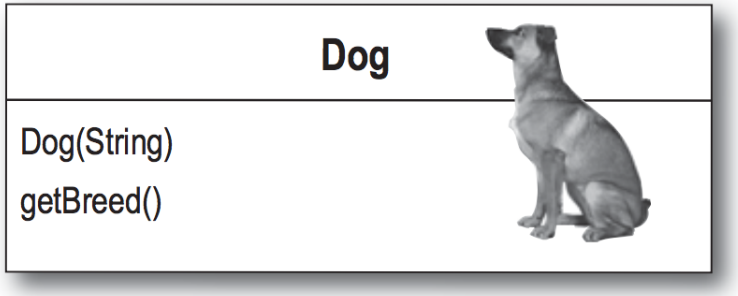
This class implements ServletContextListener, gets the context init parameters, creates the Dog, and sets the Dog as context attribute.



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 (Filters.zip)

WEB.XML

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

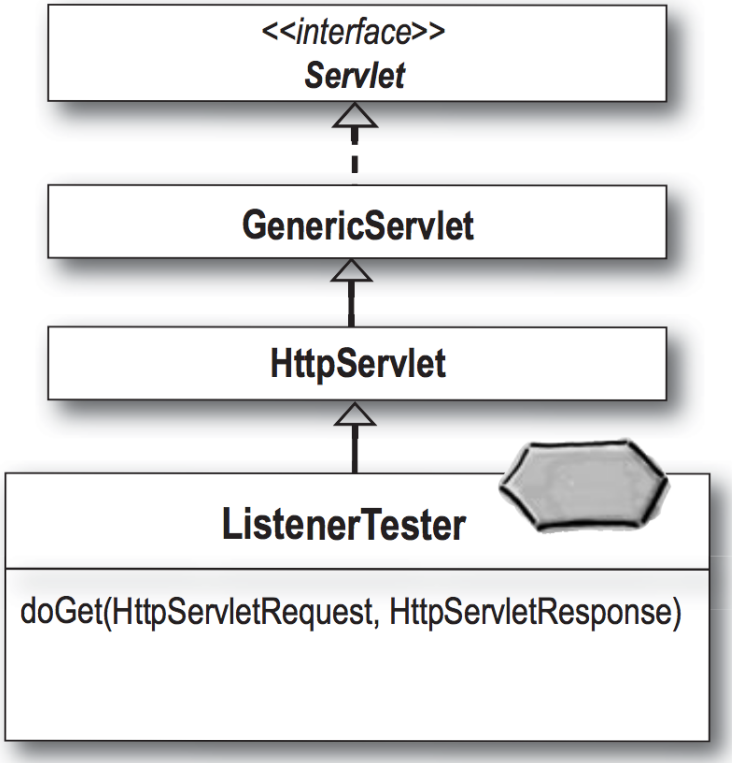
```
public class Dog {
    private String breed;

    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>");
    }
}
```

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 changed
  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