

Servlet Basics II

Sang Shin

JPassion.com

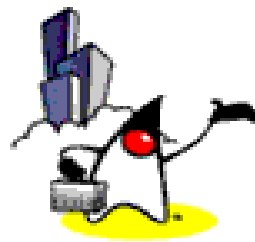
“Code with Passion”

Topics

- Servlet response: Status, Header, Body
- Servlet response status code
- Servlet scope objects
- Init parameters
- Error Handling
- Dispatcher include
- Logging

Servlet Response

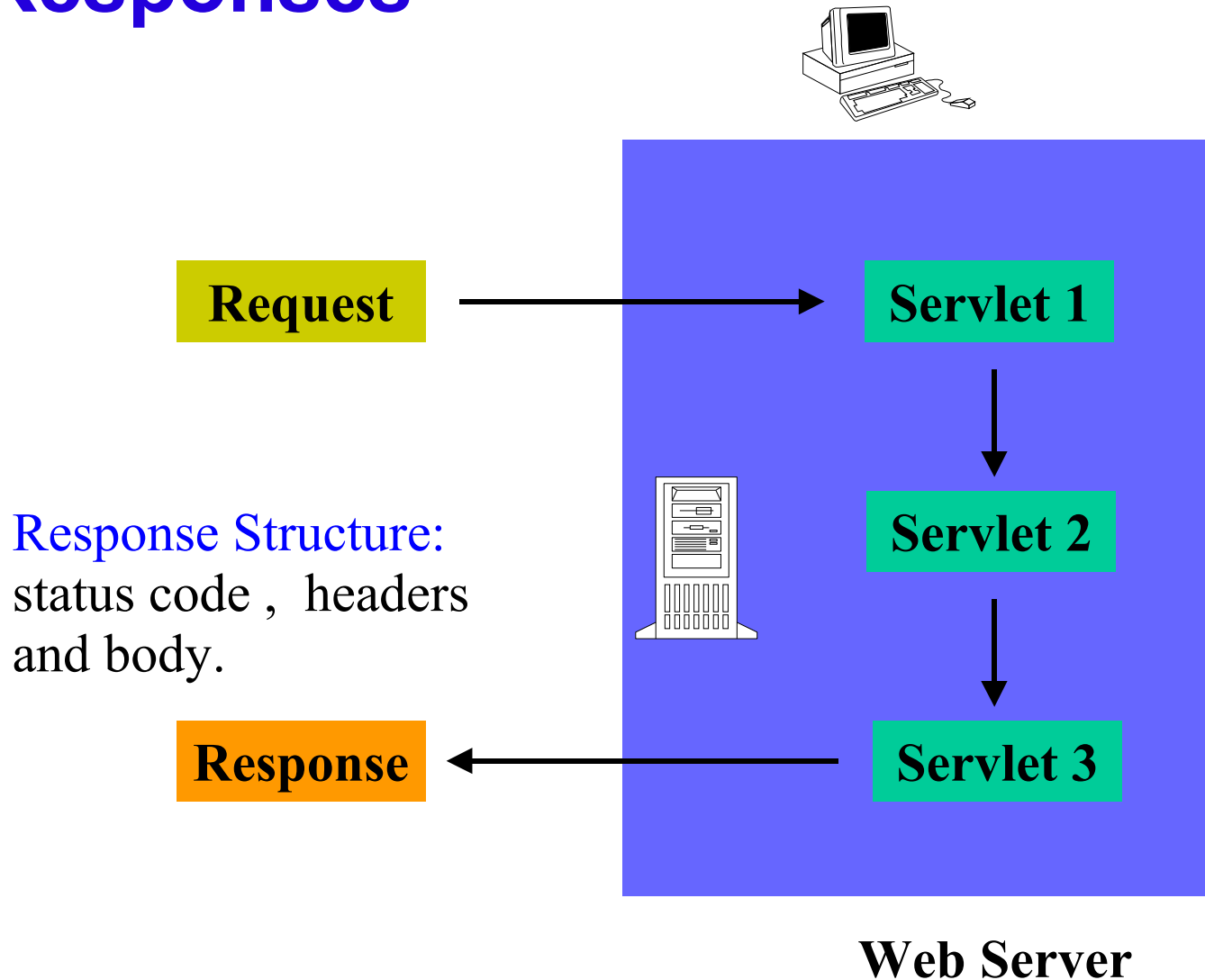
(HttpServletResponse)



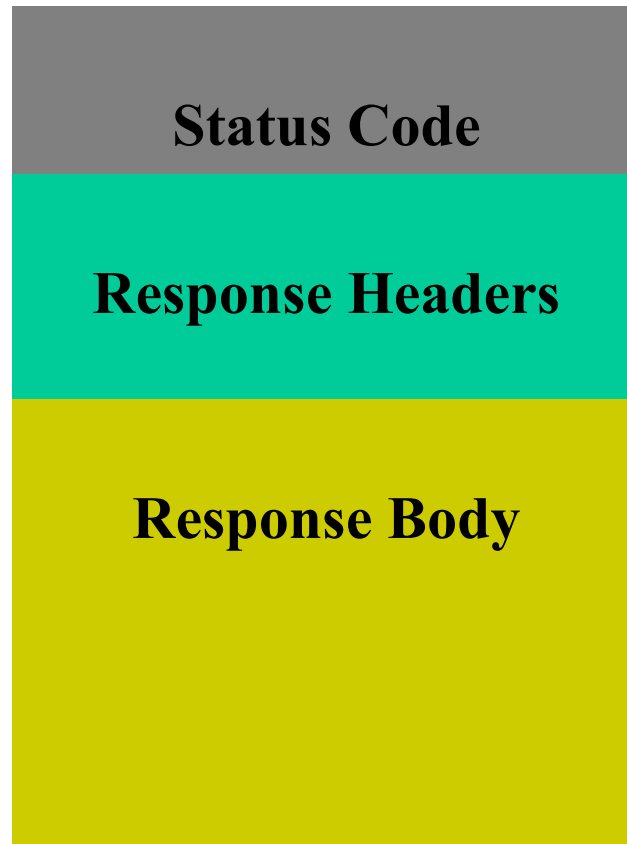
What is Servlet Response?

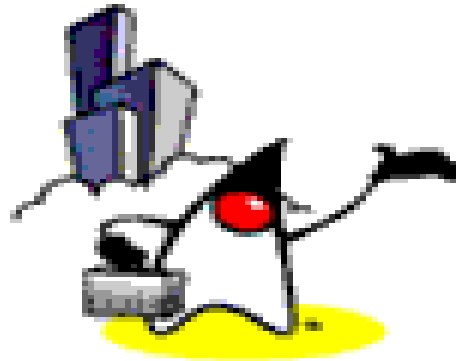
- Contains data passed from servlet to client
- All servlet responses implement `ServletResponse` interface
 - Retrieve an output stream
 - Indicate content type
 - Indicate whether to buffer output
 - Set localization information
- [`HttpServletResponse`](#) extends `ServletResponse`
 - HTTP response status code
 - Cookies

Responses



Response Structure





Status Code in Http Response

HTTP Response Status Codes

- Why do we need HTTP response status code?
 - Forward client to another page
 - Indicates resource is missing
 - Instruct browser to use cached copy

Methods for Setting HTTP Response Status Codes

- `public void setStatus(int statusCode)`
 - Status codes are defined in `HttpServletResponse`
 - Status codes are numeric fall into five general categories:
 - 100-199 Informational
 - 200-299 Successful
 - 300-399 Redirection
 - 400-499 Incomplete
 - 500-599 Server Error
 - Default status code is 200 (OK)

Example of HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```

Common Status Codes

- 200 (SC_OK)
 - Success and document follows
 - Default for servlets
- 204 (SC_No_CONTENT)
 - Success but no response body
 - Browser should keep displaying previous document
- 301 (SC_MOVED_PERMANENTLY)
 - The document moved permanently (indicated in Location header)
 - Browsers go to new location automatically

Common Status Codes

- 302 (SC_MOVED_TEMPORARILY)
 - Redirect
 - Servlets should use `sendRedirect`, not `setStatus`, when setting this header
- 401 (SC_UNAUTHORIZED)
 - Browser tried to access password-protected page without proper Authorization header
- 404 (SC_NOT_FOUND)
 - No such page

Methods for Sending Error

- Error status codes (400-599) can be used in `sendError` methods.
- `public void sendError(int code)`
 - The server may give the error special treatment
- `public void sendError(int code, String message)`
 - Wraps `message` inside small HTML document

setStatus() & sendError()

```
try {  
    returnAFile(fileName, out)  
}  
catch (FileNotFoundException e) {  
    response.setStatus(response.SC_NOT_FOUND);  
    out.println("Response body");  
}
```

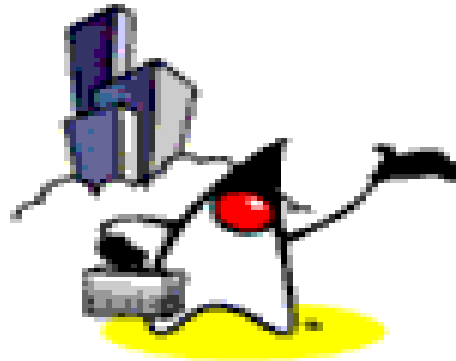
has same effect as

```
try {  
    returnAFile(fileName, out)  
}  
catch (FileNotFoundException e) {  
    response.sendError(response.SC_NOT_FOUND);  
}
```

Lab:

**Exercise 1: `sendError()`
`4007_servlet_basics2.zip`**





Header in Http Response

Why HTTP Response Headers?

- Give forwarding location
- Specify cookies
- Supply the page modification date
- Instruct the browser to reload the page after a designated interval
- Give the file size so that persistent HTTP connections can be used
- Designate the type of document being generated
- etc.

Methods for Setting Arbitrary Response Headers

- `public void setHeader( String headerName, String headerValue)`
 - Sets an arbitrary header.
- `public void setDateHeader( String name, long millisecs)`
 - Converts milliseconds since 1970 to a date string in GMT format
- `public void 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.

Methods for setting Common Response Headers

- **setContentType**
 - Sets the Content- Type header. Servlets almost always use this.
- **setContentLength**
 - Sets the Content- Length header. Used for persistent HTTP connections.
- **addCookie**
 - Adds a value to the Set- Cookie header.
- **sendRedirect**
 - Sets the Location header and changes status code.

Common HTTP 1.1 Response Headers

- Location
 - Specifies a document's new location.
 - Use `sendRedirect()` method instead of setting this directly
- Refresh
 - Specifies a delay before the browser automatically reloads a page
- Set-Cookie
 - The cookies that browser should remember
 - Don't set this header directly
 - Use `addCookie(Cookie cookie)` method instead.

Common HTTP 1.1 Response Headers (cont.)

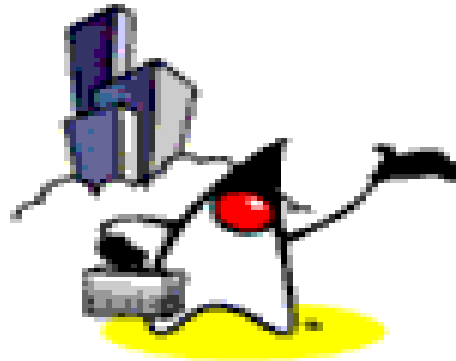
- Cache-Control (1.1) and Pragma (1.0)
 - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- Content- Encoding
 - The way document is encoded. Browser reverses this encoding before handling document.
- Content- Length
 - The number of bytes in the response. Used for persistent HTTP connections.

Common HTTP 1.1 Response Headers (cont.)

- Content-Type
 - The MIME type of the document being returned.
 - Use `setContentType()` method to set this header.
- Last-Modified
 - The time document was last changed

Refresh Sample Code

```
public class DateRefresh extends HttpServlet {  
    public void doGet(HttpServletRequest req,  
                      HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        res.setHeader("Refresh", "5");  
        out.println(new Date().toString());  
    }  
}
```

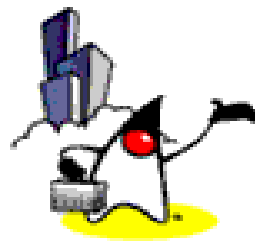


Body in Http Response

Writing a Response Body

- A servlet almost always returns a response body
- Response body could either be a `PrintWriter` or a `ServletOutputStream`
- `PrintWriter`
 - Obtained via `response.getWriter()`
 - For character-based output
- `ServletOutputStream`
 - Using `response.getOutputStream()`
 - For binary (image) data

Scope Objects



Scope Objects

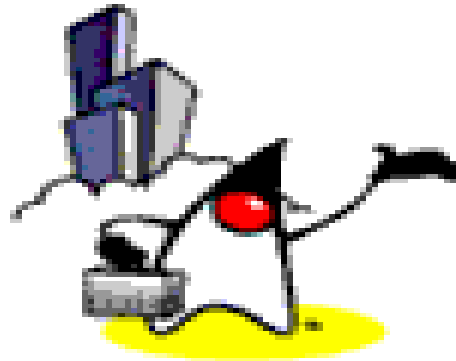
- Enables **sharing information** among collaborating web components via attributes maintained in Scope objects
 - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
 - `getAttribute()` & `setAttribute()`
- 4 Scope objects are defined
 - Web context, session, request, page

Four Scope Objects: Accessibility

- Web context (ServletContext)
 - Accessible from Web components within a Web context
- Session
 - Accessible from Web components handling a request that belongs to the session
- Request
 - Accessible from Web components handling the request
- Page
 - Accessible from JSP page that creates the object

Four Scope Objects: Class

- Web context
 - `javax.servlet.ServletContext`
- Session
 - `javax.servlet.http.HttpSession`
- Request
 - `javax.servlet.http.HttpServletRequest`
- Page
 - `javax.servlet.jsp.PageContext`



Web Context (ServletContext)

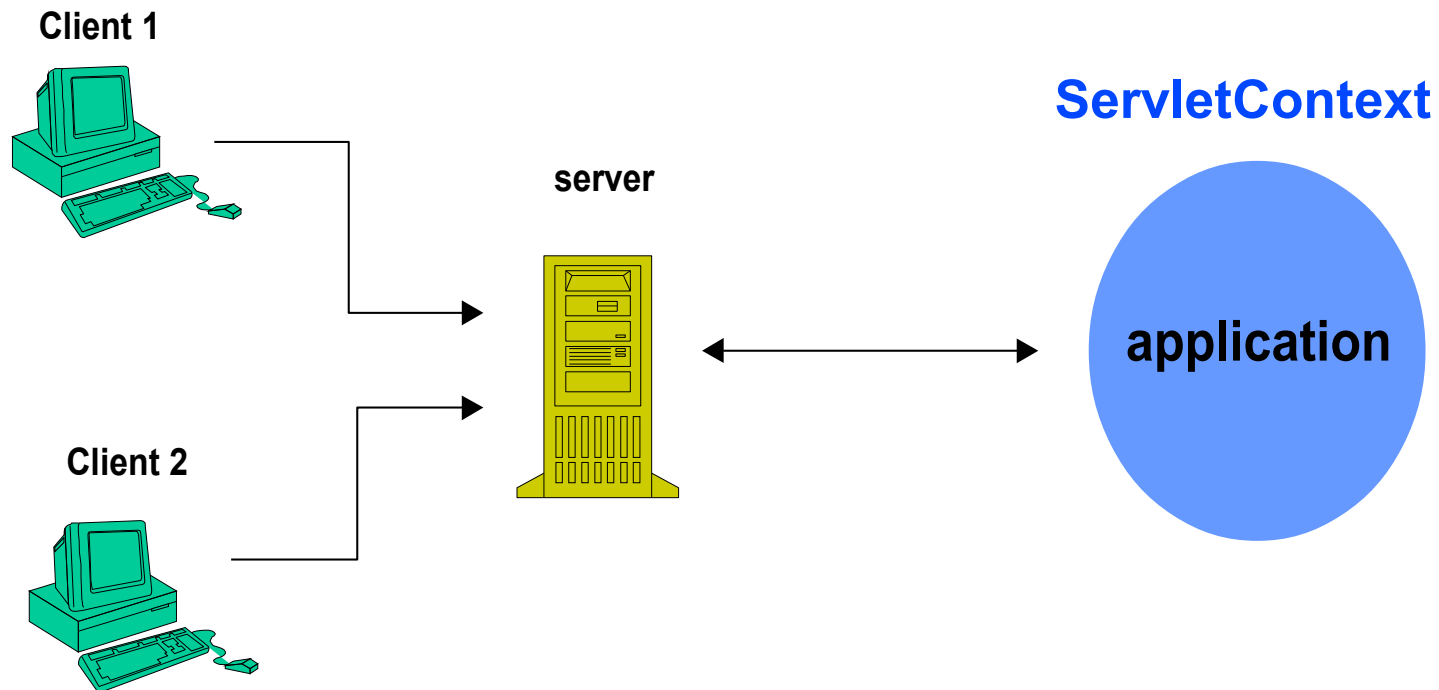
What is ServletContext For?

- Used by servets to
 - Set and get context-wide (application-wide) object-valued attributes
 - Get request dispatcher
 - To forward to or include web component
 - Access Web context-wide initialization parameters set in the web.xml file
 - Access Web resources associated with the Web context
 - Log
 - Access other misc. information

Scope of ServletContext

- Context-wide scope
 - Shared by all servlets and JSP pages within a "web application"
 - Why it is called “web application scope”
 - All servlets in BookStore web application share same ServletContext object
 - There is **one** ServletContext object per "web application"

ServletContext: Web Application Scope



How to Access ServletContext Object?

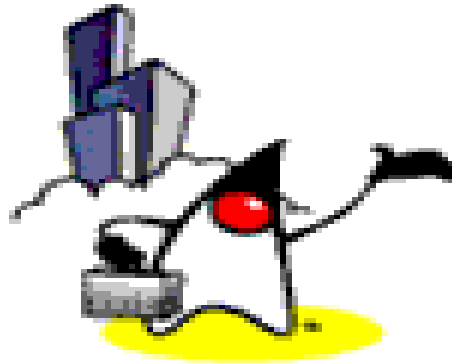
- Within your servlet code, call `getServletContext()`
- The ServletContext is contained in `ServletConfig` object, which the Web server provides to a servlet when the servlet is initialized
 - `init (ServletConfig servletConfig)` in Servlet interface

Example: Getting Attribute Value from ServletContext

```
public class CatalogServlet extends HttpServlet {
    private BookDB bookDB;

    public void init() throws ServletException {
        // Get context-wide attribute value from
        // ServletContext object
        bookDB = (BookDB) getServletContext().
                        getAttribute("bookDB");

        if (bookDB == null) throw new
            UnavailableException("Couldn't get database.");
    }
}
```



Session

(HttpSession)

**We will talk more on
HTTPSession
later in “Session Tracking”**

Why HttpSession?

- Need a mechanism to **maintain client state** across a series of requests from a same user (or originating from the same browser) over some period of time
 - Example: Online shopping cart
- Yet, HTTP is stateless
- HttpSession maintains client state
 - Used by Servlets to set and get the values of session scope attributes

How to Get HttpSession?

- via getSession() method of a Request object (HttpServletRequest)

Example: HttpSession

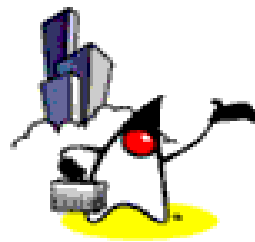
```
public class CashierServlet extends HttpServlet {  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        // Get the user's session and shopping cart  
        HttpSession session = request.getSession();  
        ShoppingCart cart =  
            (ShoppingCart) session.getAttribute("cart");  
  
        ...  
        // Determine the total price of the user's books  
        double total = cart.getTotal();  
    }  
}
```

Lab:

Exercise 2: Scope Objects
4007_servlet_basics2.zip



Init Parameters



Setting Context Init Parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" ...>
  <context-param>
    <param-name>city</param-name>
    <param-value>Seoul</param-value>
  </context-param>
  <context-param>
    <param-name>age</param-name>
    <param-value>22</param-value>
  </context-param>
  <servlet>
    <display-name>GreetingServlet</display-name>
    <servlet-name>GreetingServlet</servlet-name>
    <servlet-class>servlets.GreetingServlet</servlet-
class>
  </servlet>
```

Reading Context Init Parameters

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();

    // then write the data of the response
    String username = request.getParameter("username");

    if ((username != null) && (username.length() > 0)) {
        out.println("<h2>Hello, " + username + "!</h2>");
        out.println("<h2>You live in " +
getServletContext().getInitParameter("city") + "!</h2>");
        out.println("<h2>Your age is " +
getServletContext().getInitParameter("age") + "!</h2>");
    }
}
```

Setting Servlet Init Parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" ...>
  <servlet>
    <display-name>GreetingServlet</display-name>
    <servlet-name>GreetingServlet</servlet-name>
    <servlet-class>servlets.GreetingServlet</servlet-class>
    <init-param>
      <param-name>greeting1</param-name>
      <param-value>hello</param-value>
    </init-param>
  </servlet>
  <servlet>
    <display-name>GreetingServlet2</display-name>
    <servlet-name>GreetingServlet2</servlet-name>
    <servlet-class>servlets.GreetingServlet2</servlet-class>
    <init-param>
      <param-name>greeting2</param-name>
      <param-value>goodbye</param-value>
    </init-param>
  </servlet>
</web-app>
```

Reading Servlet Init Parameters

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();

    // then write the data of the response
    String username = request.getParameter("username");

    if ((username != null) && (username.length() > 0)) {
        out.println("<h2>Hello, " + username + "!</h2>");
        out.println("<h2>You live in " +
getServletContext().getInitParameter("city") + "!</h2>");
        out.println("<h2>Your age is " +
getServletContext().getInitParameter("age") + "!</h2>");
        out.println("<h2>Your greeting message is " +
getInitParameter("greeting3") + "!</h2>");
    }
}
```

Lab:

Exercise 3: Init Parameters 4007_servlet_basics2.zip





Handling Errors

Handling Errors

- Web container generates default error page
- You can specify custom default page to be displayed instead
- Steps to handle errors
 - Create appropriate error html pages for error conditions
 - Modify the web.xml accordingly

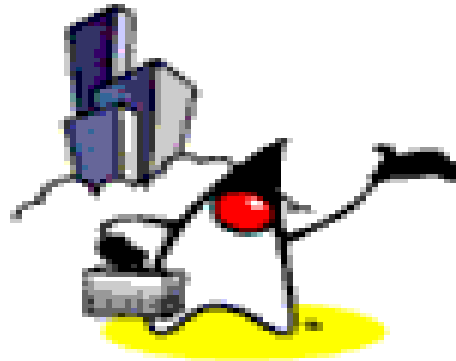
Example: Setting Error Pages in web.xml

```
<error-page>
  <exception-type>
    exception.BookNotFoundException
  </exception-type>
  <location>/errorpage1.html</location>
</error-page>
<error-page>
  <exception-type>
    exception.BooksNotFoundException
  </exception-type>
  <location>/errorpage2.html</location>
</error-page>
<error-page>
  <exception-type>exception.OrderException</exception-type>
  <location>/errorpage3.html</location>
</error-page>
```

Lab:

Exercise 4: Handling errors 4007_servlet_basics2.zip





RequestDispatcher

Example: Getting and Using RequestDispatcher Object

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

    HttpSession session = request.getSession(true);
    ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

    // set headers and buffer size before accessing the Writer
    response.setContentType("text/html");
    response.setBufferSize(8192);
    PrintWriter out = response.getWriter();

    // then write the response
    out.println("<html>" +
               "<head><title>" + messages.getString("TitleBookDescription") +
               "</title></head>");

    // Get the dispatcher; it gets the banner to the user
    RequestDispatcher dispatcher =
        session.getServletContext().getRequestDispatcher("/banner");

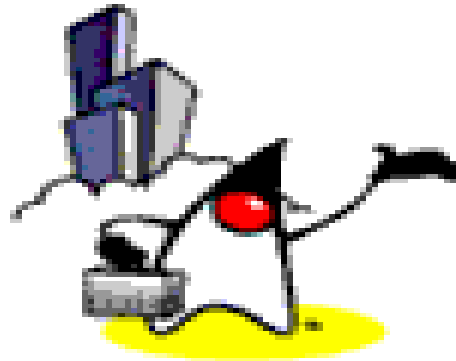
    if (dispatcher != null)
        dispatcher.include(request, response);

    ...
}
```

Lab:

Exercise 5: Dispatcher
4007_servlet_basics2.zip





Logging

Example: Logging

```
public void doGet (HttpServletRequest request,  
                  HttpServletResponse response)  
    throws ServletException, IOException {  
  
    ...  
    getServletContext().log("Life is good!");  
    ...  
    getServletContext().log("Life is bad!", someException);  
}
```

Lab:

Exercise 6: Logging
4007_servlet_basics2.zip



Code with Passion!
JPassion.com

