

Filter

Filter

#Filter #Controller #JavaEE

Kiều Trọng Khánh

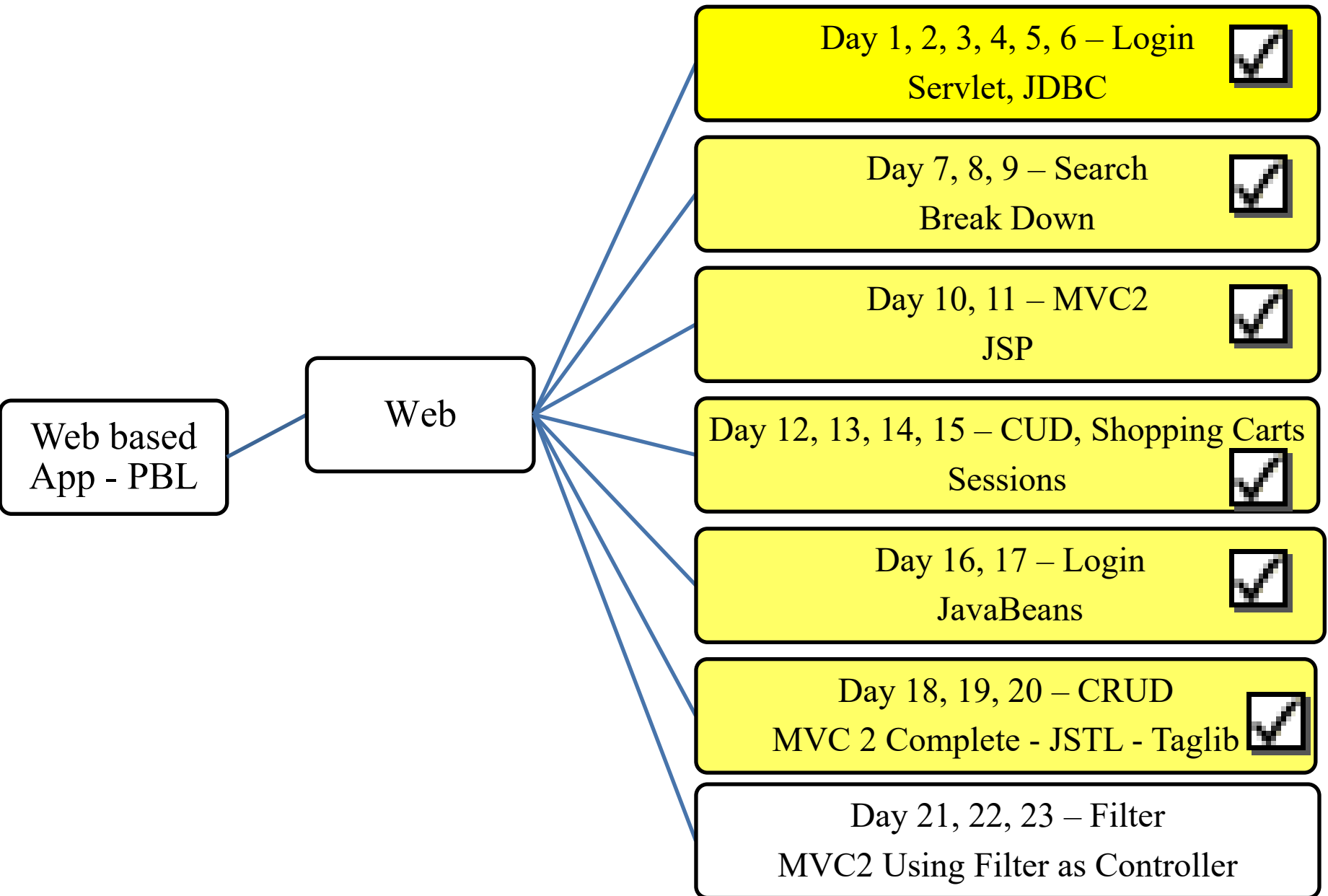
Review

- **How to remove all java code in JSP (View)?
Complete the MVC 2 Design Pattern with View
– JSTL**
- **How to build the data grid tag library using in JSP?**
 - **Tag Libraries**
 - Model
 - Classical, Simple, and Handles
 - How to implement the custom Tag Lib and use it in JSP

Objectives

- **How to build application using MVC2 Pattern using Filter as Controller?**
 - Filter
 - Filter Chain
 - Using Filter as Controller in MVC2 Design Pattern

Objectives



Filter

Requirements

- **Preprocess request and Postprocess response in web application**
- **Build the Project CRUD, Online Shopping applying MVC2 pattern with Filter as Controller**

Filter

Overview

- **Are components that add functionality to the request and response processing of a Web Application**
 - **Intercept** the requests and response that flow between a client and a Servlet/JSP.
 - Supports **dynamic modification of requests and responses** between client and web applications.
 - Dynamically **access incoming requests** from the user before the servlet processes the request
 - **Access the outgoing response** from the web resources before it reaches the user
- **Categorized** according to the **services** they provide to the web applications
- **Resides** in the **web container** along with the web applications
- Was introduced as a Web component in Java servlet specification version 2.3

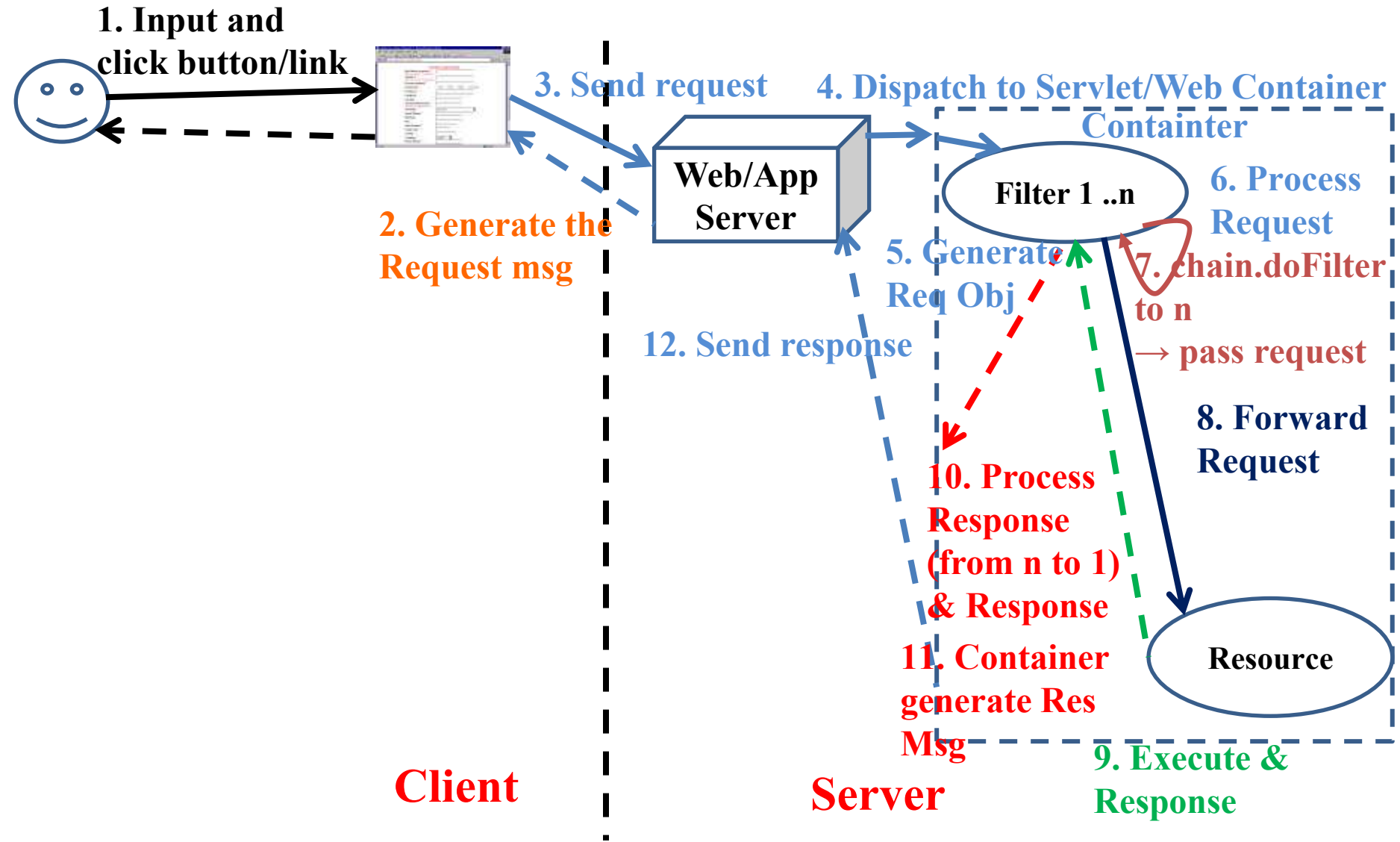
Filter

Overview

- **Are components that add functionality to the request and response processing of a Web Application**
 - **Intercept** the requests and response that flow between a client and a Servlet/JSP.
 - Supports **dynamic modification of requests and responses** between client and web applications.
 - Dynamically **access incoming requests** from the user before the servlet processes the request
 - **Access the outgoing response** from the web resources before it reaches the user
- **Categorized** according to the **services** they provide to the web applications
- **Resides** in the **web container** along with the web applications
- Was introduced as a Web component in Java servlet specification version 2.3

Filter

Interactive Filter Model



Filter

Usage

- Authorize request
- Altering request headers and modify data
- Modify response headers and data
- Authenticating the user
- Comprising files
- Encrypting data
- Converting images
- Logging and auditing filters
- Filters that trigger resource access events

Filter

Benefits - Advantages

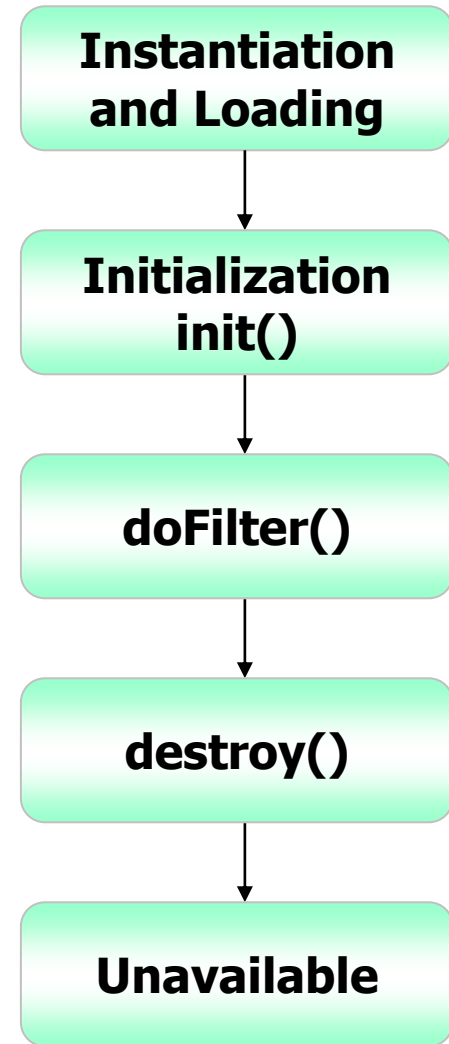
- **Optimization** of the **time** taken to send a response
- **Compression** of the **content** size before sending
- **Optimization** of the **bandwidth**
- **Security**
- **Identify** the **type of request** coming from the Web client, such as HTTP and FTP, and invoke the Servlet that needs to process the request.
- **Retrieve** the user information from the request parameters to authenticate the user.
- **Validate** a client using Servlet filters before the client accesses the Servlet.
- **Identify** the **information** about the MIME types and other header contents of the request.
- **Facilitate** a **Servlet** to **communicate** with the **external resources**.
- **Intercept** responses and compress it before sending the response to the client

Filter

Life Cycle

- Working of Filter

- The filter intercepts the request from a user to the servlet
- The filter then provides customized services
- The filter sends the serviced response or request to the appropriate destination



Filter

API

- **Creates and handles** the functionalities of a filter
- Contains **three interfaces**
 - Filter Interface, FilterConfig Interface, FilterChain Interface
- **Filter Interface**
 - Must be implemented to create a filter class **extends javax.servlet.Filter**
 - An object performs filtering tasks on the request and the response

Methods	Descriptions
init	<ul style="list-style-type: none">- public void init(FilterConfig fg);- Called by the servlet container to initialize the filter- Called only once- Must complete successfully before the filter is asked to do any filtering work
doFilter	<ul style="list-style-type: none">- public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException- Called by the container each time a request or response is processed- Then examines the request/response headers & customizes them as per the requirements- Passed the request/response through the FilterChain object to the next entity in the chain
destroy	<ul style="list-style-type: none">- public void destroy();- Called by the servlet container to inform the filter that its service is no more required- Called only once.

Filter Configuration

- In Web Deployment Descriptor

```
<web-app>
```

```
....
```

```
<filter>
```

```
  <filter-name>Name of Filters</filter-name>
```

```
  <filter-class>implemented Filter Class</filter-class>
```

```
  [<init-param>
```

```
    <param-name>parameter name</param-name>
```

```
    <param-value>value </param-value>
```

```
  </init-param>]
```

```
</filter>
```

```
<filter-mapping>
```

```
  <filter-name>FilterName</filter-name>
```

```
  <url-pattern>/context</url-pattern>
```

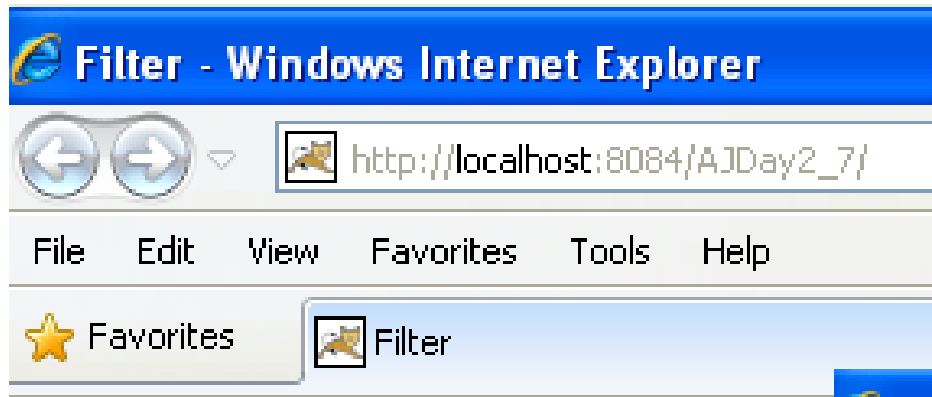
```
</filter-mapping>
```

```
....
```

```
</web-app>
```

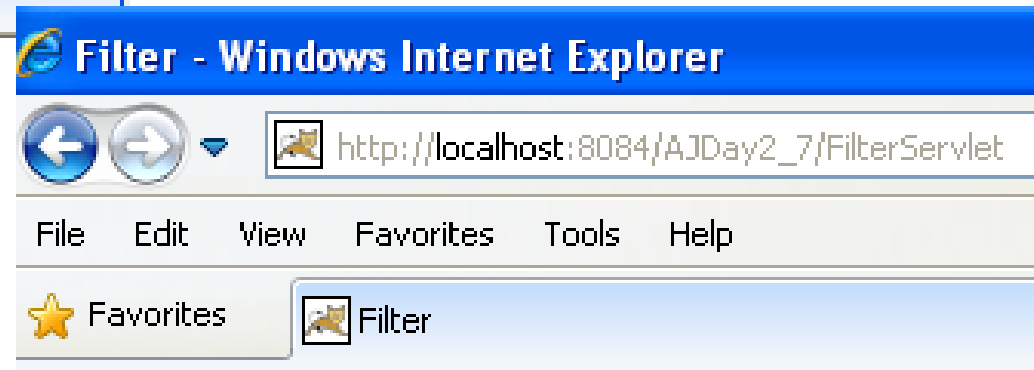
Filter Example

- Building the web application shows as the following GUI in sequence



Filter Demo

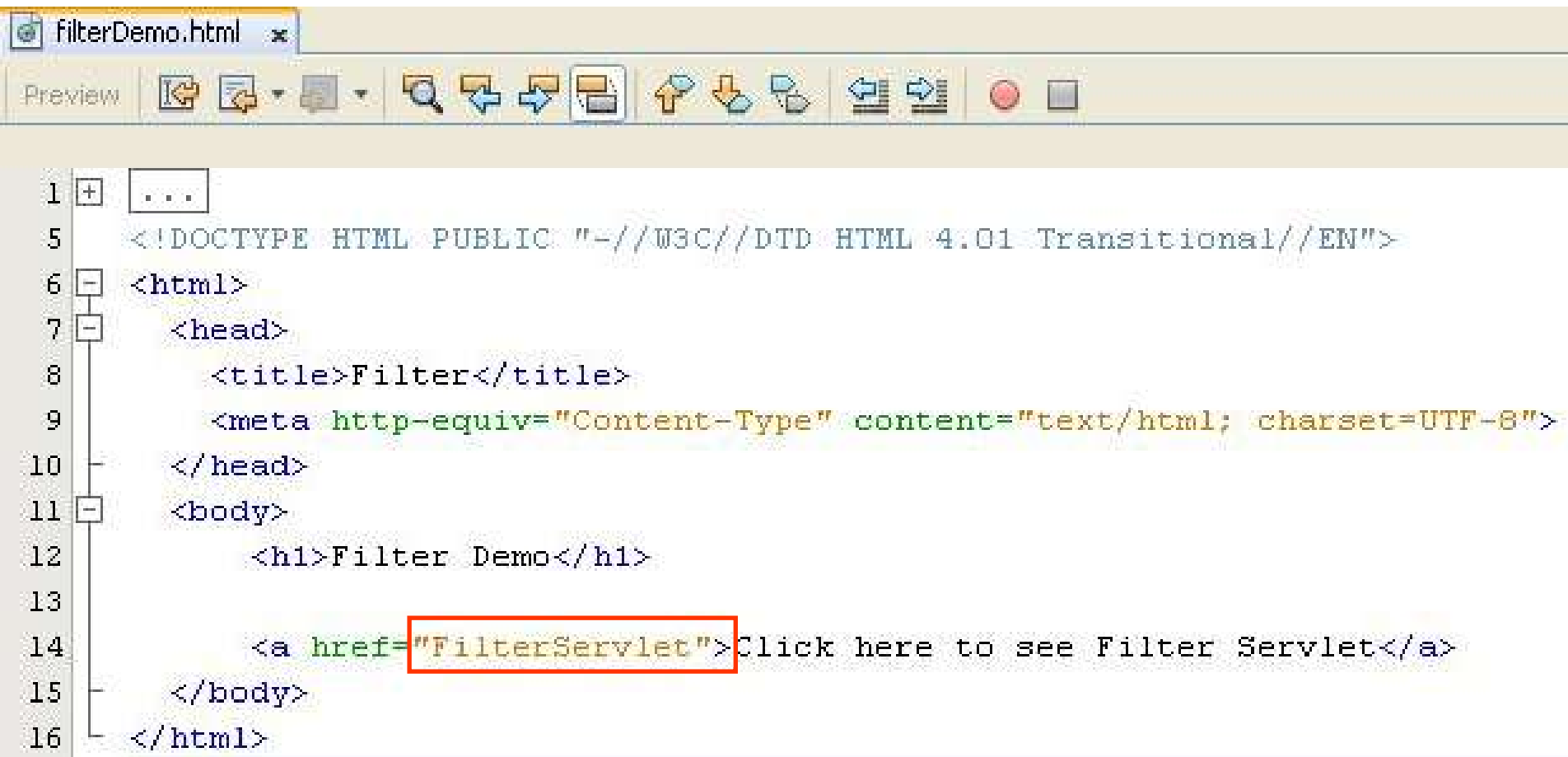
[Click here to see Filter Servlet](#)



Filter Demo

KEY is First Filter

Filter Example



```
1  ...
5  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
6  <html>
7    <head>
8      <title>Filter</title>
9      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
10   </head>
11   <body>
12     <h1>Filter Demo</h1>
13
14     <a href="FilterServlet">Click here to see Filter Servlet</a>
15   </body>
16 </html>
```

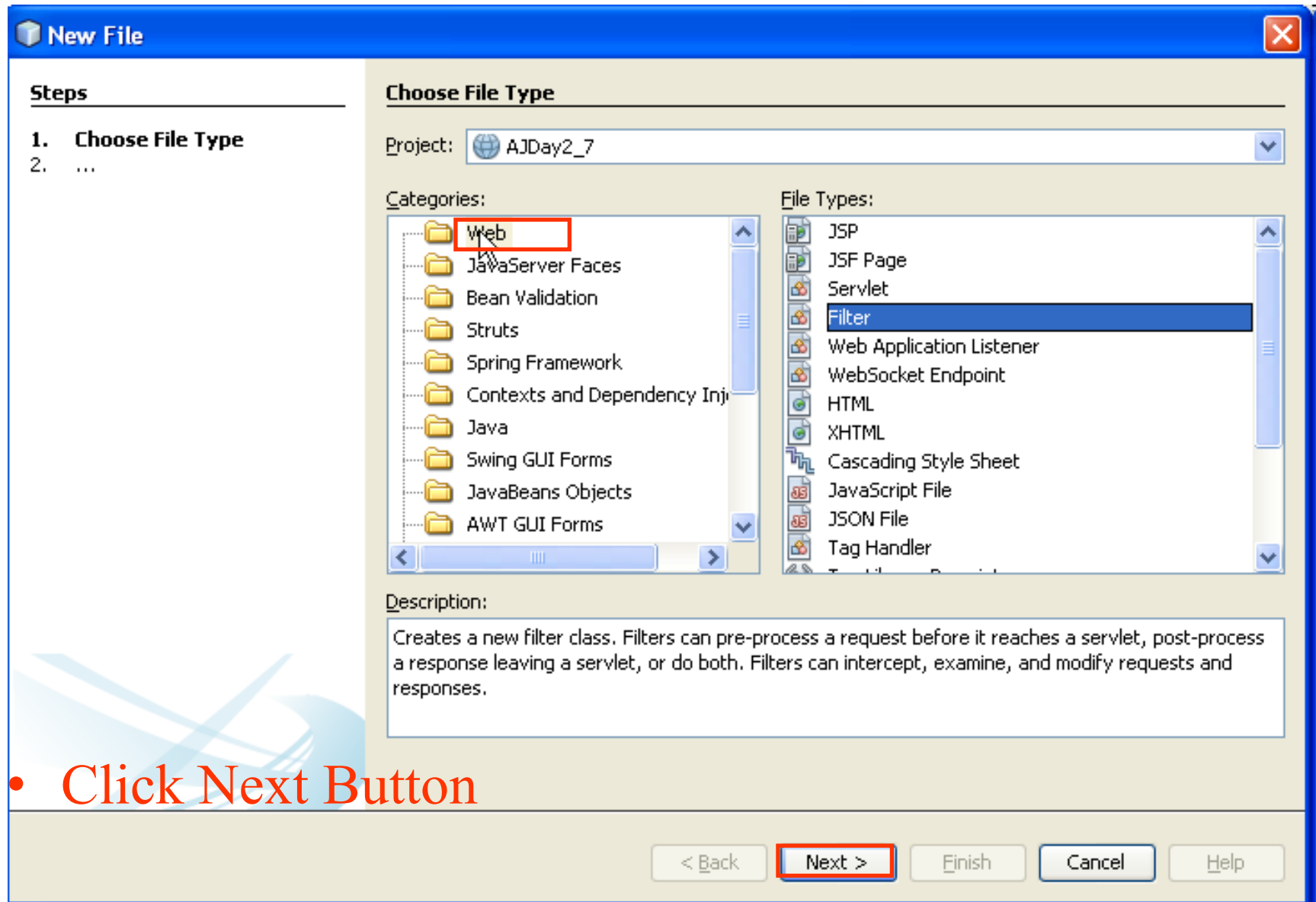
Filter Example

```

FilterServlet.java x
17  * @author Trong Khanh
18  */
19  public class FilterServlet extends HttpServlet {
20
21      /**...*/
22
23      protected void processRequest(HttpServletRequest request, HttpServletResponse response)
24      throws ServletException, IOException {
25          response.setContentType("text/html;charset=UTF-8");
26          PrintWriter out = response.getWriter();
27          try {
28              out.println("<html>");
29              out.println("<head>");
30              out.println("<title>Filter</title>");
31              out.println("</head>");
32              out.println("<body>");
33              out.println("<h1>Filter Demo</h1>");
34
35              String test = (String) request.getAttribute("KEY");
36              out.println("KEY is " + test);
37
38              out.println("</body>");
39              out.println("</html>");
40          } finally {
41              out.close();
42          }
43      }
44  }

```


Filter Example



- Click Next Button

Filter Example

New Filter

Steps

1. Choose File Type
2. **Name and Location**
3. Configure Filter Deployment
4. Filter Init Parameters

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

☐ Wrap Request and Response Objects

< Back **Next >** Finish Cancel Help

Fill your filter name

Fill/choose package name

- Click Next Button

Filter Example

New Filter

Steps

1. Choose File Type
2. Name and Location
3. **Configure Filter Deployment**
4. Filter Init Parameters

Configure Filter Deployment

Register the Filter with the application by giving the Filter an internal name. Describe when the Filter is invoked by listing the HTTP request path patterns or Servlets to which the Filter applies. Order this Filter's mappings relative to any other Filter invocation.

Class Name:

Filter Name:

Filter Mappings:

Filter name	Applies to
FirstFilter	/*

Buttons: New..., Edit..., Delete, Move Up, Move Down

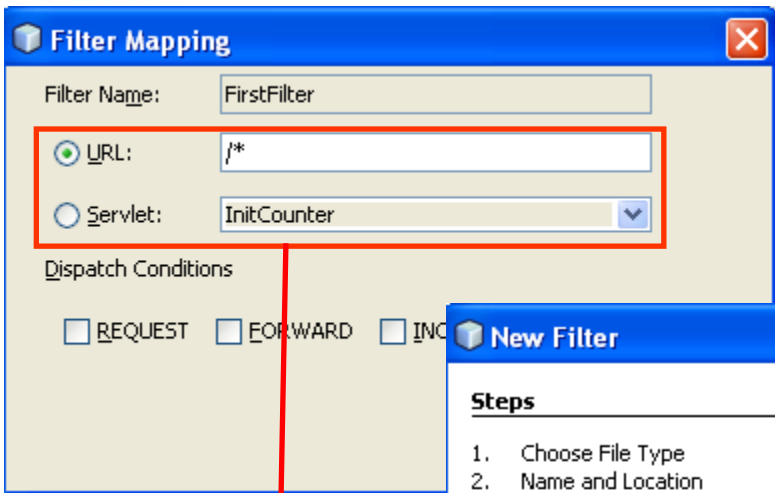
Navigation: < Back, Next >, **Finish**, Cancel, Help

Apply filter

Edit the Apply filter

- Click Edit Button to apply Filter the selected Servlet
- Otherwise, click Finish Button

Filter Example



Filter Mapping

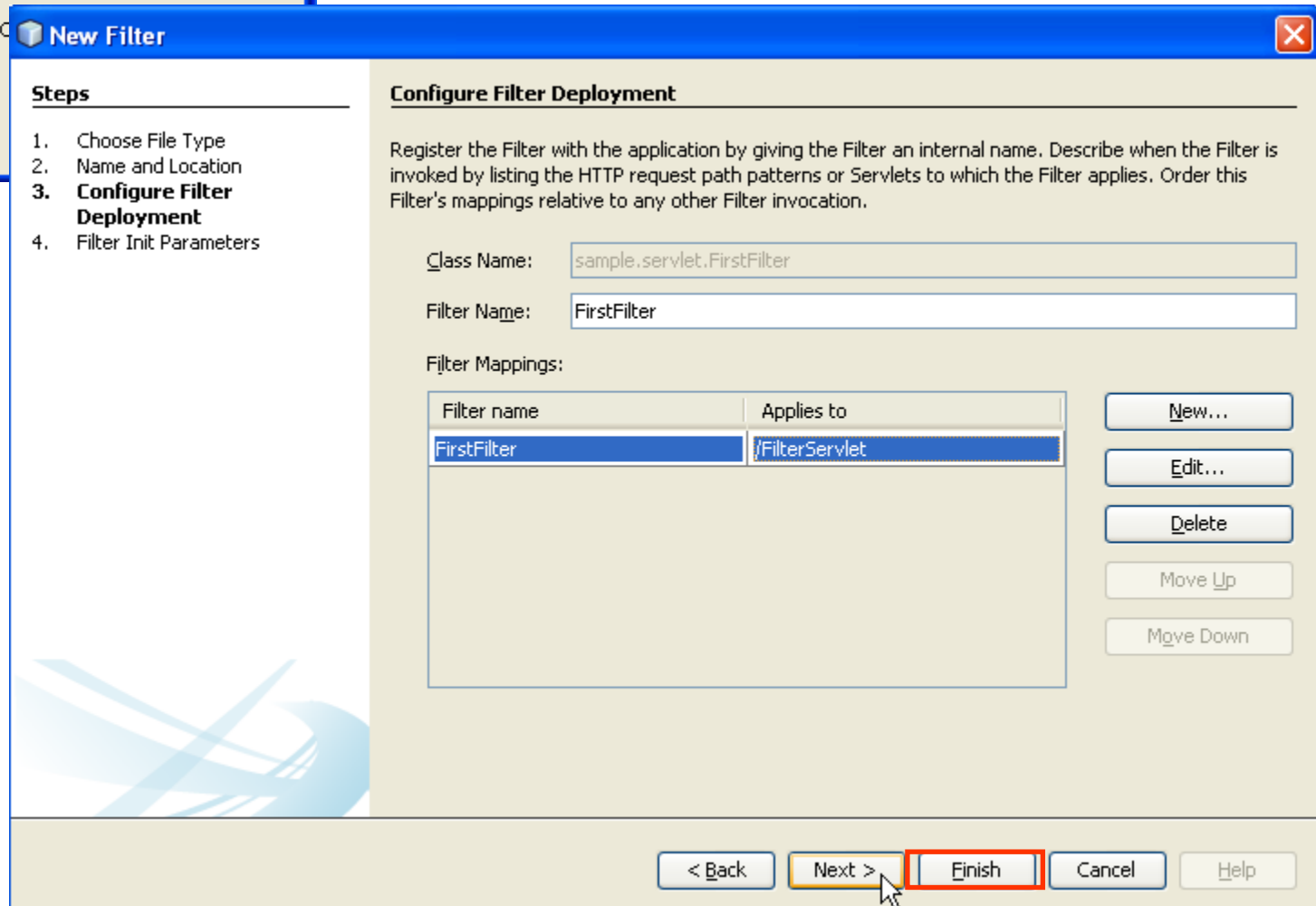
Filter Name: FirstFilter

☒ URL: /*

☐ Servlet: InitCounter

Dispatch Conditions

☐ REQUEST ☐ FORWARD ☐ INCLUDE



New Filter

Steps

1. Choose File Type
2. Name and Location
3. **Configure Filter Deployment**
4. Filter Init Parameters

Configure Filter Deployment

Register the Filter with the application by giving the Filter an internal name. Describe when the Filter is invoked by listing the HTTP request path patterns or Servlets to which the Filter applies. Order this Filter's mappings relative to any other Filter invocation.

Class Name: sample.servlet.FirstFilter

Filter Name: FirstFilter

Filter Mappings:

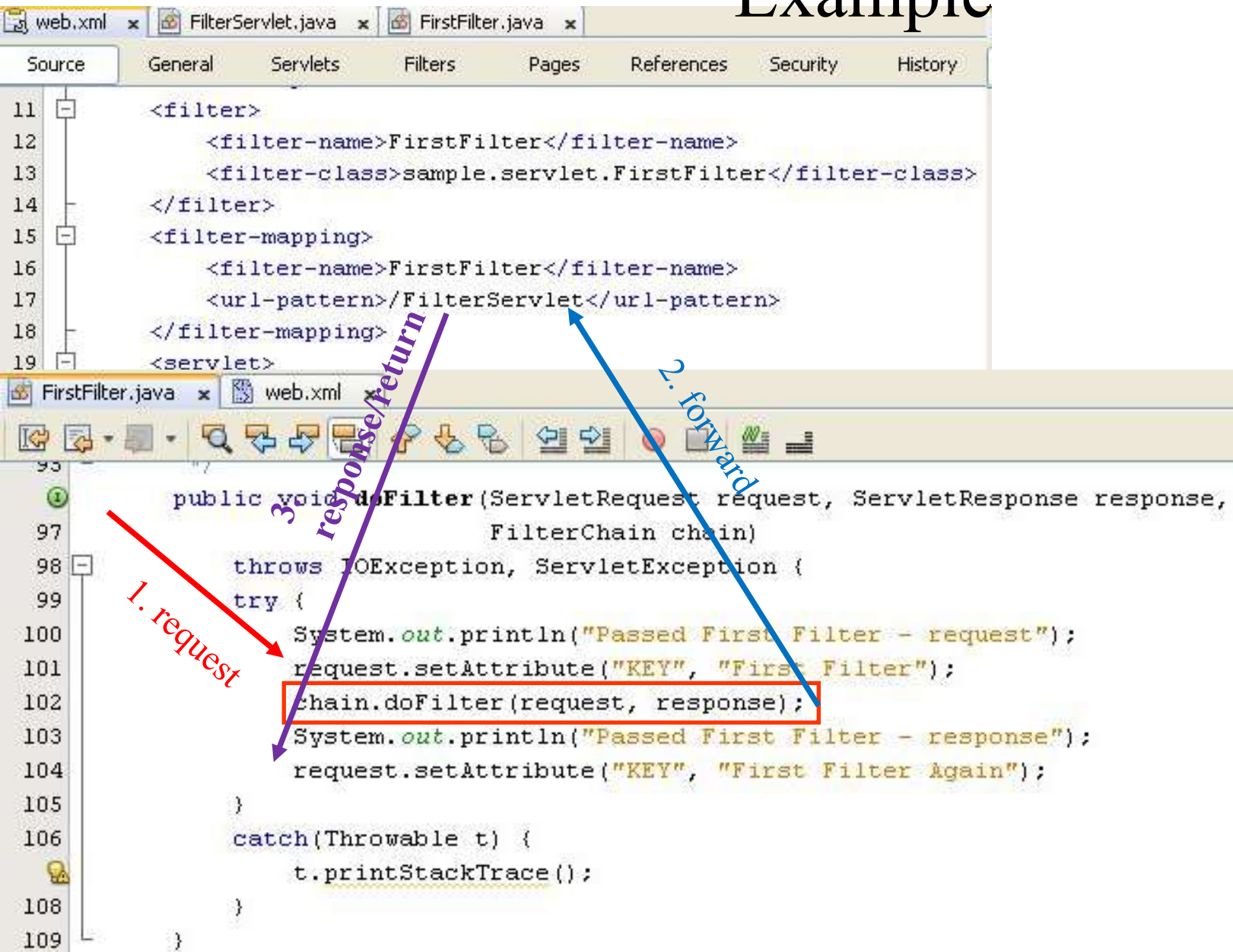
Filter name	Applies to
FirstFilter	/FilterServlet

Buttons: New..., Edit..., Delete, Move Up, Move Down

Bottom Buttons: < Back, Next >, Finish, Cancel, Help

Select the URL and typing the URL string, or Select the Servlet and choose the approximate Servlet in combo box

Filter Example



The screenshot shows an IDE with two tabs: `web.xml` and `FirstFilter.java`. The `web.xml` tab is active, showing the configuration for the `FirstFilter`.

```

11 <filter>
12     <filter-name>FirstFilter</filter-name>
13     <filter-class>sample.servlet.FirstFilter</filter-class>
14 </filter>
15 <filter-mapping>
16     <filter-name>FirstFilter</filter-name>
17     <url-pattern>/FilterServlet</url-pattern>
18 </filter-mapping>
19 <servlet>

```

The `FirstFilter.java` tab is also visible, showing the implementation of the `doFilter` method. The method is annotated with three steps:

1. request (indicated by a red arrow pointing to the `request` parameter)
2. forward (indicated by a blue arrow pointing to the `chain.doFilter(request, response);` line, which is highlighted with a red box)
3. response/return (indicated by a purple arrow pointing to the `response` parameter)

```

95 public void doFilter(ServletRequest request, ServletResponse response,
96                     FilterChain chain)
97     throws IOException, ServletException {
98     try {
99         System.out.println("Passed First Filter - request");
100         request.setAttribute("KEY", "First Filter");
101         chain.doFilter(request, response);
102         System.out.println("Passed First Filter - response");
103         request.setAttribute("KEY", "First Filter Again");
104     }
105     catch(Throwable t) {
106         t.printStackTrace();
107     }
108 }
109 }

```

Filter Example

Output

Apache Tomcat 7.0.27.0 x Apache Tomcat 7.0.27.0 Log x AJDay2_7 (run-deploy) x

```

thg 10 30, 2013 8:08:38 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter

```

Output

Apache Tomcat 7.0.27.0 x Apache Tomcat 7.0.27.0 Log x AJDay2_7 (run-deploy) x

```

Passed First Filter - request
Passed First Filter - response

```

web.xml x FilterServlet.java x FirstFilter.java x

Source

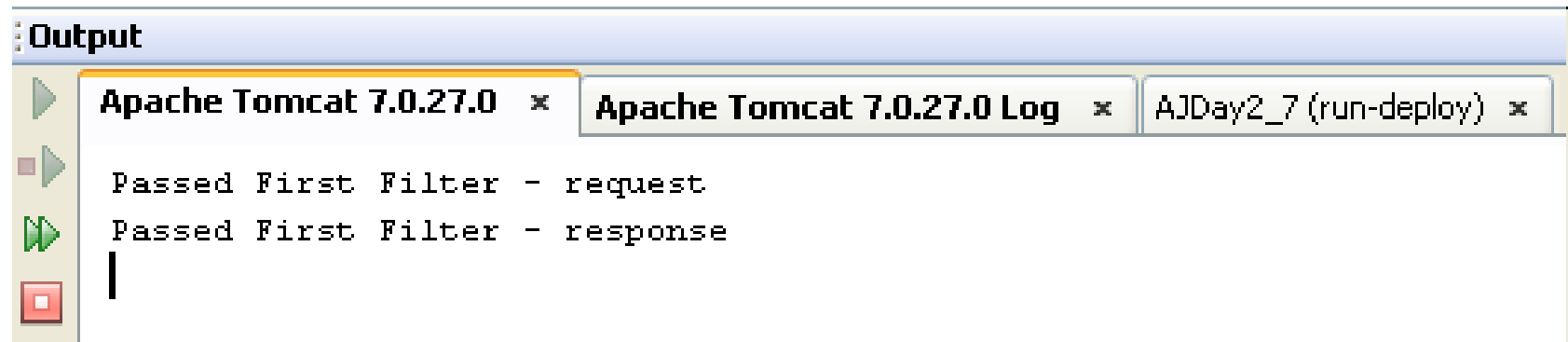
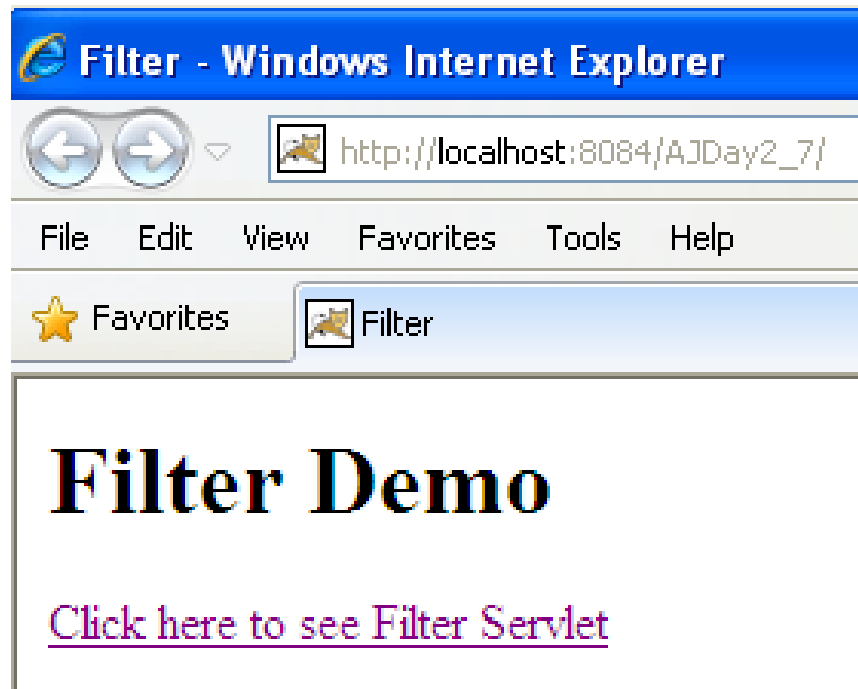
General Servlets Filters Pages References Security History

```

11 <filter>
12     <filter-name>FirstFilter</filter-name>
13     <filter-class>sample.servlet.FirstFilter</filter-class>
14 </filter>
15 <filter-mapping>
16     <filter-name>FirstFilter</filter-name>
17     <url-pattern>/*</url-pattern>
18 </filter-mapping>

```

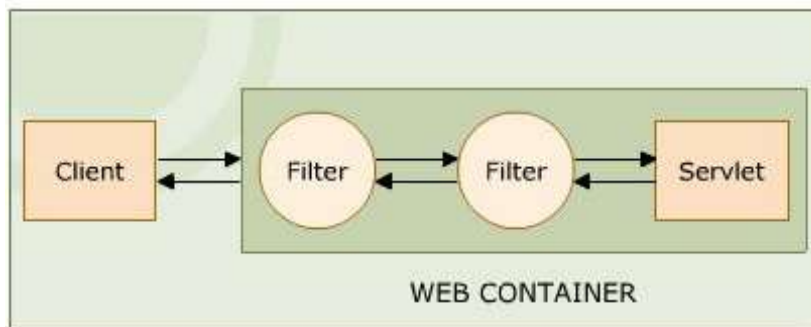
Filter Example



Filter Chain

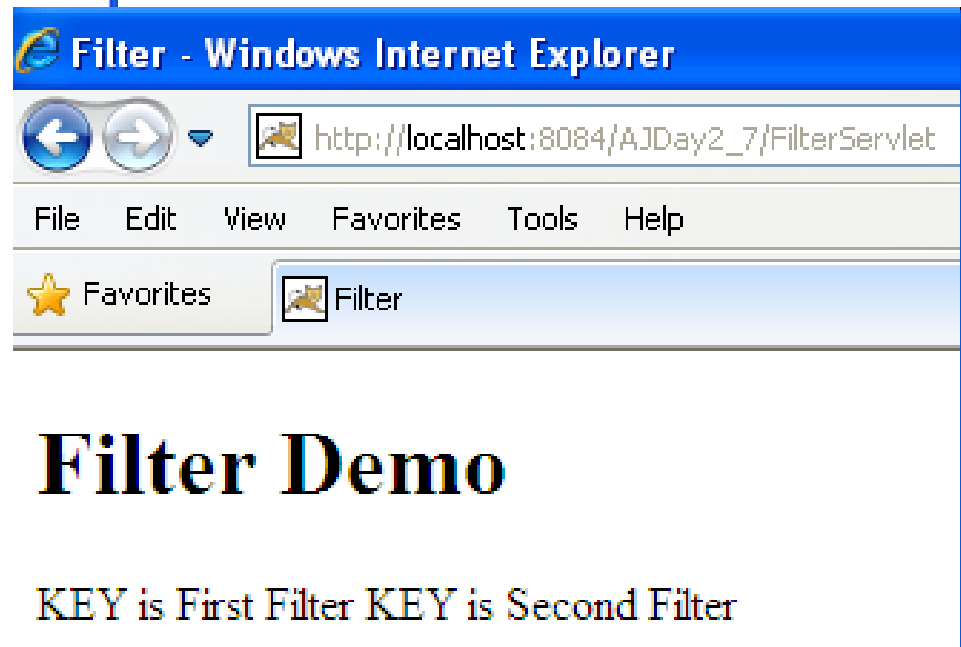
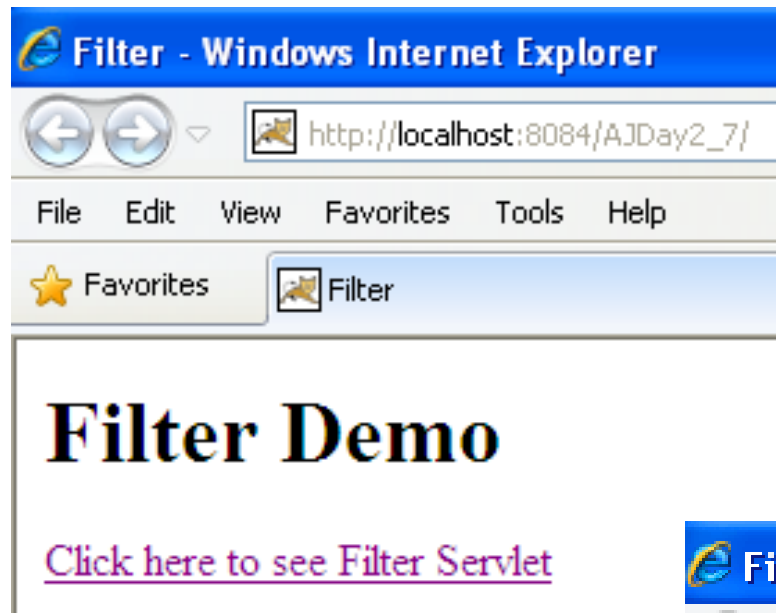
Definition

- There can be **more than one filter** between the user and the endpoint – Invoke a **series of filters**
- A request or a response is **passed through one** filter to the **next** in the filter chain. So each request and response has to be serviced by each filter forming a filter chain
- If the Calling filter is last filter, will invoke web resource
- **FilterChain Interface**
 - Provides an object through the web container
 - The object invokes the next filter in a filter chain starting from the first filter from a particular end. If the calling filter is the last filter in the chain, it will invoke the web resource, such as JSP and servlet.
 - Only implement doFilter() method.
 - Forces the next filter in the chain to be invoked



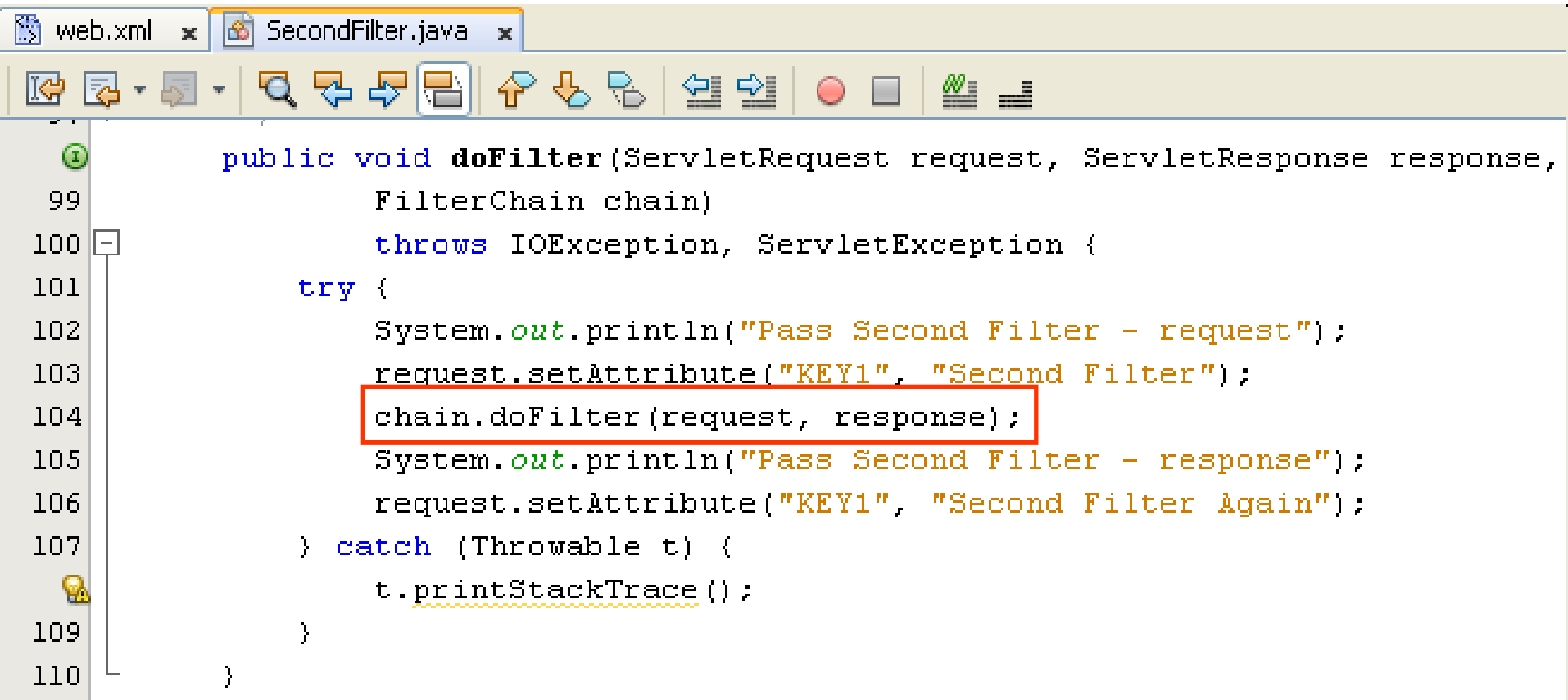
Filter Chain

Filter Chain Example



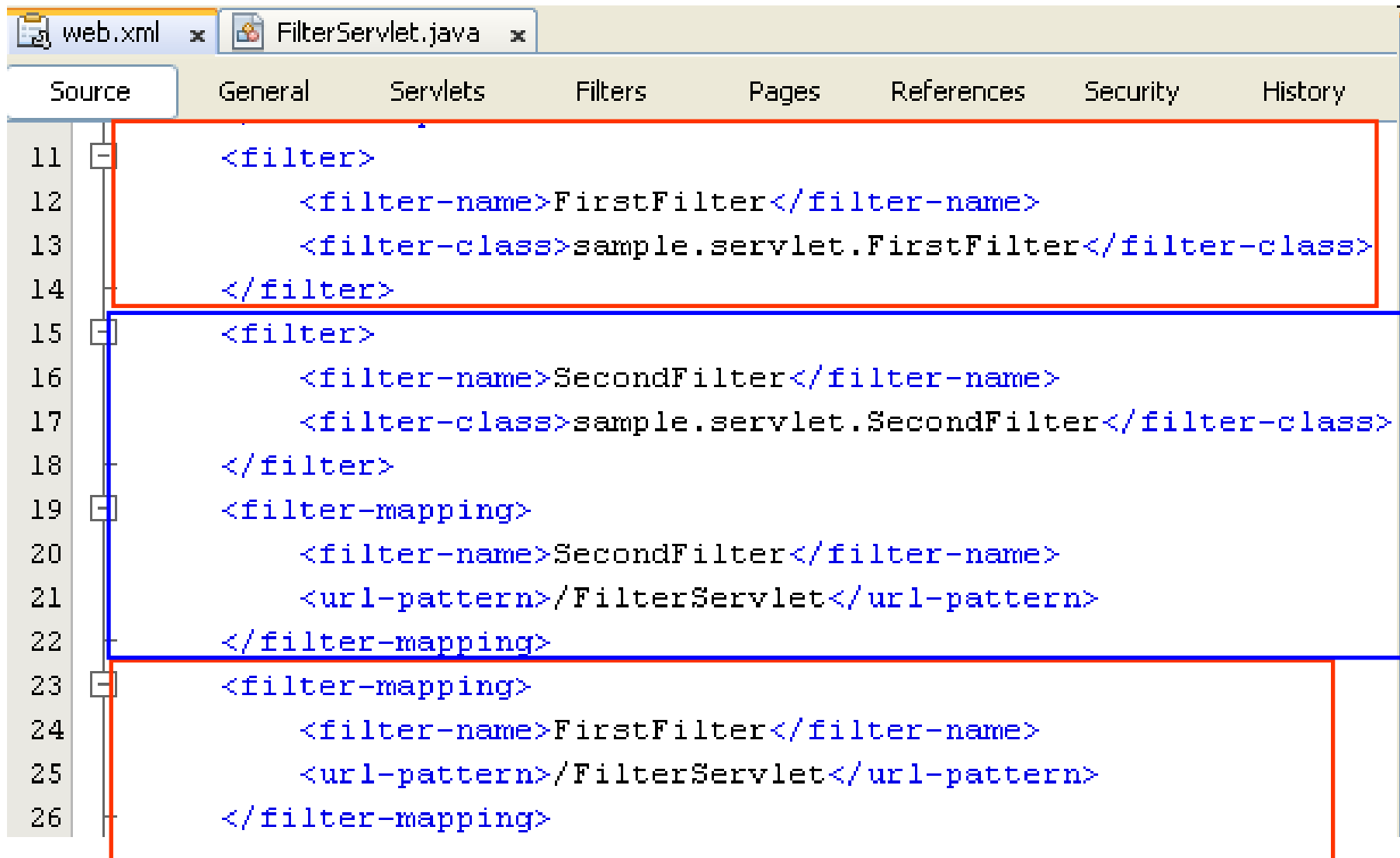
Filter Chain

Example



```
web.xml x SecondFilter.java x
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
    throws IOException, ServletException {
    try {
        System.out.println("Pass Second Filter - request");
        request.setAttribute("KEY1", "Second Filter");
        chain.doFilter(request, response);
        System.out.println("Pass Second Filter - response");
        request.setAttribute("KEY1", "Second Filter Again");
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

Filter Chain Example



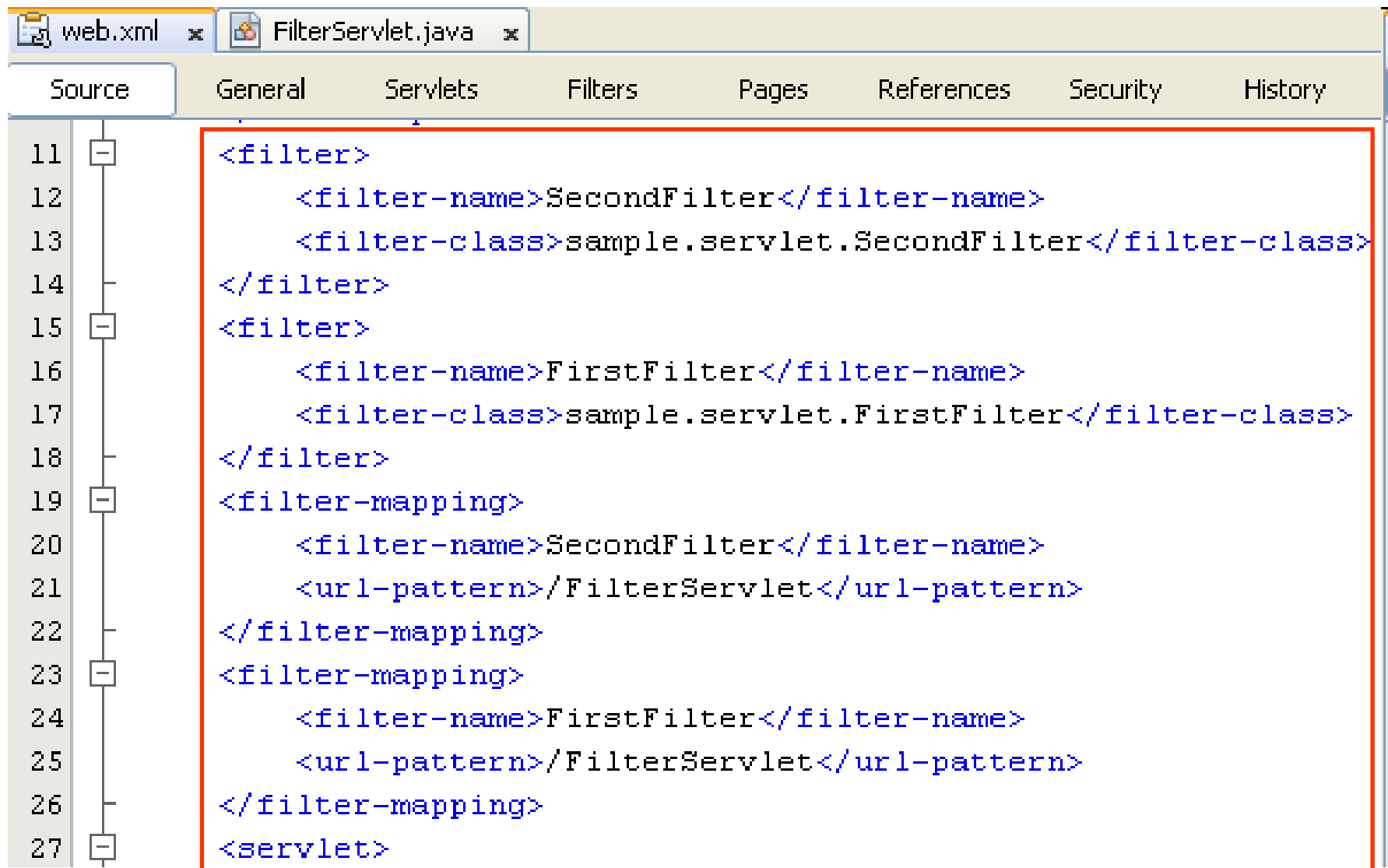
The screenshot shows an IDE with two tabs: 'web.xml' and 'FilterServlet.java'. The 'web.xml' tab is active, displaying XML configuration for filters and filter-mappings. The code is as follows:

```

11 <filter>
12     <filter-name>FirstFilter</filter-name>
13     <filter-class>sample.servlet.FirstFilter</filter-class>
14 </filter>
15 <filter>
16     <filter-name>SecondFilter</filter-name>
17     <filter-class>sample.servlet.SecondFilter</filter-class>
18 </filter>
19 <filter-mapping>
20     <filter-name>SecondFilter</filter-name>
21     <url-pattern>/FilterServlet</url-pattern>
22 </filter-mapping>
23 <filter-mapping>
24     <filter-name>FirstFilter</filter-name>
25     <url-pattern>/FilterServlet</url-pattern>
26 </filter-mapping>
  
```

The configuration defines two filters: 'FirstFilter' and 'SecondFilter'. It then defines two 'filter-mapping' entries for the URL pattern '/FilterServlet'. The first mapping uses 'SecondFilter', and the second mapping uses 'FirstFilter', creating a filter chain where the request is first processed by 'SecondFilter' and then by 'FirstFilter'.

Filter Chain Example



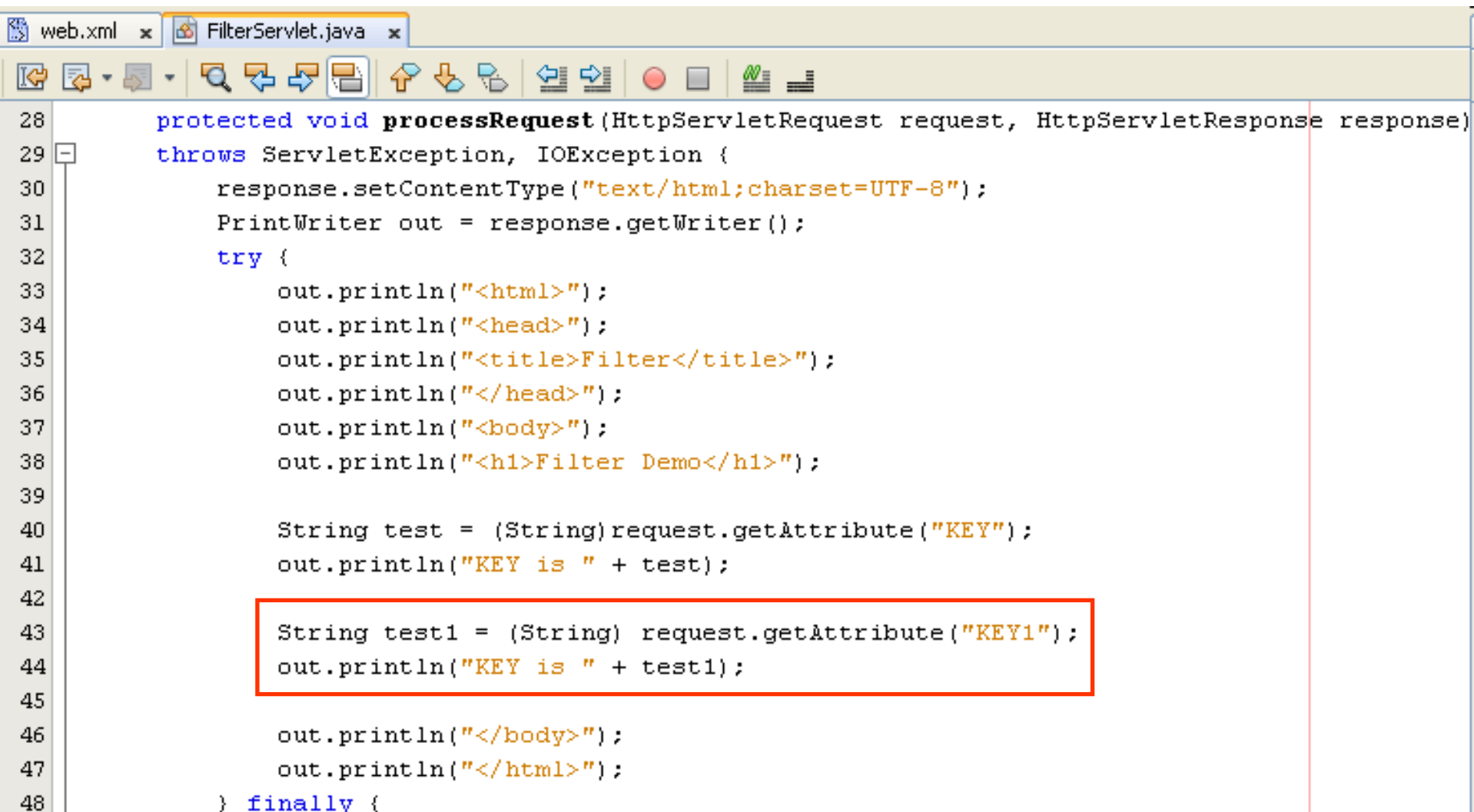
The screenshot shows an IDE window with two tabs: 'web.xml' and 'FilterServlet.java'. The 'web.xml' tab is active, and the 'Source' editor is displaying the XML configuration. The configuration defines two filters, 'SecondFilter' and 'FirstFilter', and two filter-mappings for the 'FilterServlet'. The filter-mappings are ordered such that 'SecondFilter' is applied first, followed by 'FirstFilter', creating a filter chain. The entire XML content is enclosed in a red rectangular border.

```

11 <filter>
12     <filter-name>SecondFilter</filter-name>
13     <filter-class>sample.servlet.SecondFilter</filter-class>
14 </filter>
15 <filter>
16     <filter-name>FirstFilter</filter-name>
17     <filter-class>sample.servlet.FirstFilter</filter-class>
18 </filter>
19 <filter-mapping>
20     <filter-name>SecondFilter</filter-name>
21     <url-pattern>/FilterServlet</url-pattern>
22 </filter-mapping>
23 <filter-mapping>
24     <filter-name>FirstFilter</filter-name>
25     <url-pattern>/FilterServlet</url-pattern>
26 </filter-mapping>
27 <servlet>

```

Filter Chain Example



```
web.xml x FilterServlet.java x
28 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
29 throws ServletException, IOException {
30     response.setContentType("text/html;charset=UTF-8");
31     PrintWriter out = response.getWriter();
32     try {
33         out.println("<html>");
34         out.println("<head>");
35         out.println("<title>Filter</title>");
36         out.println("</head>");
37         out.println("<body>");
38         out.println("<h1>Filter Demo</h1>");
39
40         String test = (String)request.getAttribute("KEY");
41         out.println("KEY is " + test);
42
43         String test1 = (String) request.getAttribute("KEY1");
44         out.println("KEY is " + test1);
45
46         out.println("</body>");
47         out.println("</html>");
48     } finally {
```

Filter Chain Example

Output

Apache Tomcat 7.0.27.0 x

Apache Tomcat 7.0.27.0 Log x

AJDay2_7 (run-deploy) x

```
thg 10 30, 2013 8:11:16 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
thg 10 30, 2013 8:11:16 SA org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

Apache Tomcat 7.0.27.0 x

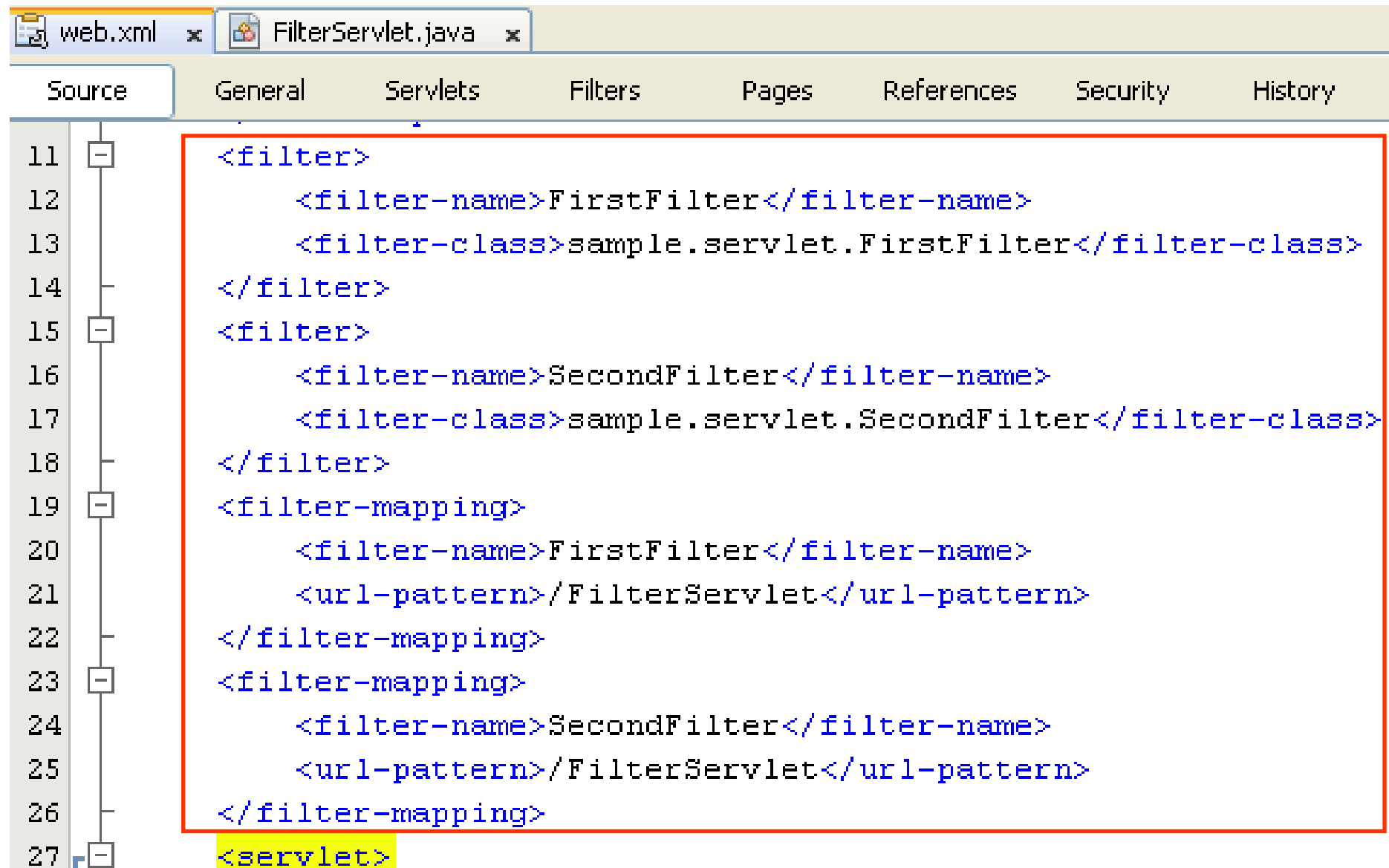
Apache Tomcat 7.0.27.0 Log x

AJDay2_7 (run-deploy) x

```
Pass Second Filter - request
Passed First Filter - request
Passed First Filter - response
Pass Second Filter - response
```

Filter Chain

Example – Change position



The screenshot shows an IDE with two tabs: web.xml and FilterServlet.java. The web.xml tab is active, and the code is displayed in the Source view. The code defines a filter chain with two filters, FirstFilter and SecondFilter, and maps them to the FilterServlet. The FirstFilter is mapped to the URL pattern /FilterServlet, and the SecondFilter is also mapped to the same URL pattern. The code is highlighted with a red border.

```

11 <filter>
12     <filter-name>FirstFilter</filter-name>
13     <filter-class>sample.servlet.FirstFilter</filter-class>
14 </filter>
15 <filter>
16     <filter-name>SecondFilter</filter-name>
17     <filter-class>sample.servlet.SecondFilter</filter-class>
18 </filter>
19 <filter-mapping>
20     <filter-name>FirstFilter</filter-name>
21     <url-pattern>/FilterServlet</url-pattern>
22 </filter-mapping>
23 <filter-mapping>
24     <filter-name>SecondFilter</filter-name>
25     <url-pattern>/FilterServlet</url-pattern>
26 </filter-mapping>
27 <servlet>

```

Filter Chain Example

Output

Apache Tomcat 7.0.27.0 x Apache Tomcat 7.0.27.0 Log x AJDay2_7 (run-deploy) x

```
thg 10 30, 2013 8:06:32 SA org.apache.catalina.core.ApplicationContext log
INFO: FirstFilter:Initializing filter
thg 10 30, 2013 8:06:32 SA org.apache.catalina.core.ApplicationContext log
INFO: SecondFilter:Initializing filter
```

Output

▶ Apache Tomcat 7.0.27.0 x ▶ Apache Tomcat 7.0.27.0 Log x ▶ AJDay2_7 (run-deploy) x

```
Passed First Filter - request
Pass Second Filter - request
Pass Second Filter - response
Passed First Filter - response
```


Filter Chain

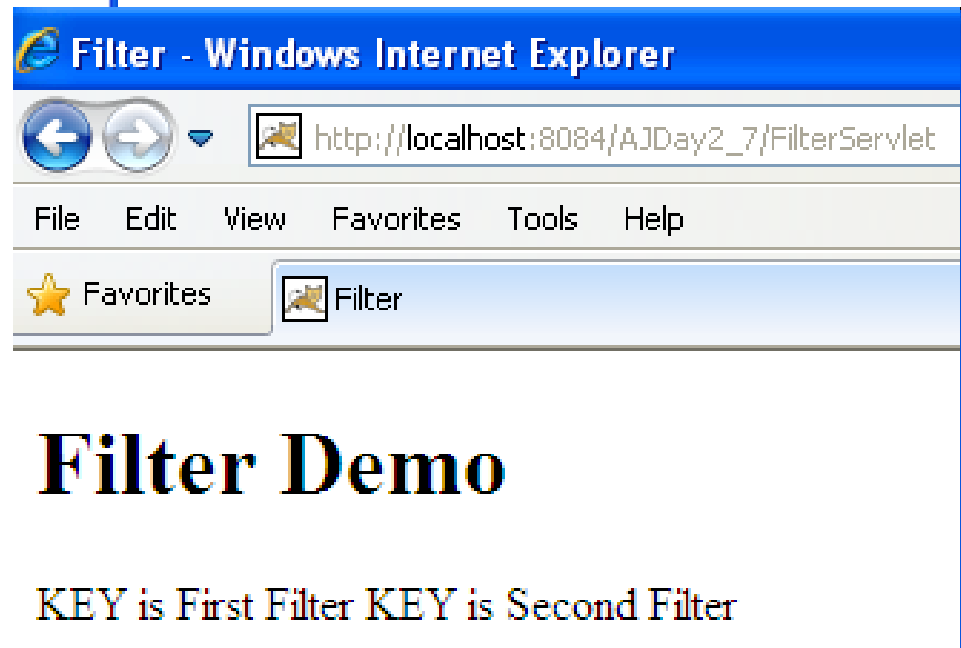
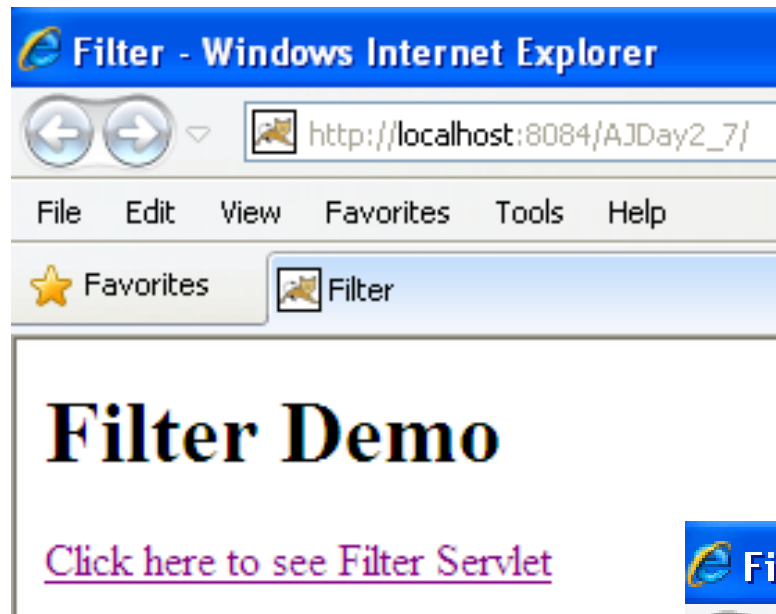
Why need a Wrapper Class

```

SecondFilter.java
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
    throws IOException, ServletException {
    try {
        System.out.println("Pass Second Filter - request");
        request.setAttribute("KEY1", "Second Filter");
        chain.doFilter(request, response);
        System.out.println("Pass Second Filter - response");
        request.setAttribute("KEY1", "Second Filter Again");
        PrintWriter out = response.getWriter();
        out.println("<br/>The slide is licensed to KhanhKT");
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
  
```

Filter Chain

Why need a Wrapper Class

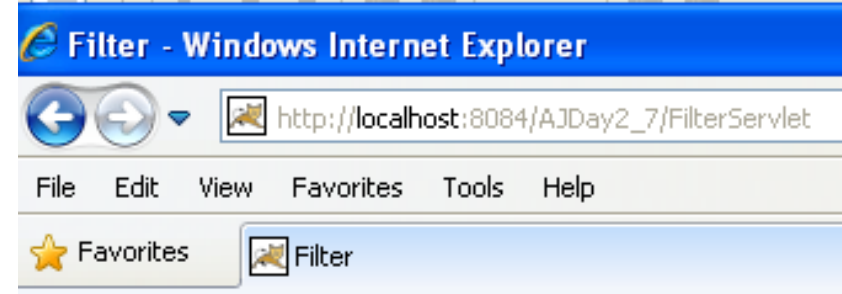


Filter Chain

Why need a Wrapper Class

```

28  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
29  throws ServletException, IOException {
30      response.setContentType("text/html;charset=UTF-8");
31      PrintWriter out = response.getWriter();
32      try {
33          out.println("<html>");
34          out.println("<head>");
35          out.println("<title>Filter</title>");
36          out.println("</head>");
37          out.println("<body>");
38          out.println("<h1>Filter Demo</h1>");
39
40          String test = (String)request.getAttribute("key");
41          out.println("KEY is " + test);
42
43          String test1 = (String) request.getAttribute("key");
44          out.println("KEY is " + test1);
45
46          out.println("</body>");
47          out.println("</html>");
48      } finally {
49          //out.close();
50      }
51  }
    
```



Filter Demo

KEY is First Filter KEY is Second Filter

The slide is licensed to KhanhKT

Filter Chain

Wrapper Class

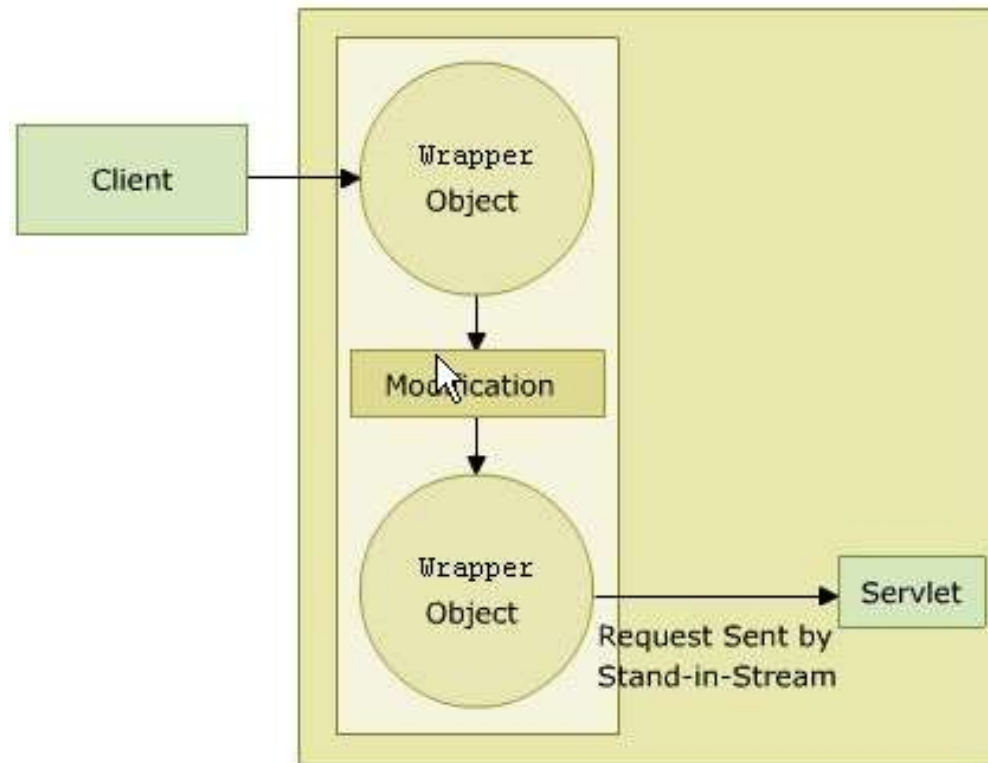
- To **modify or intercept** the request or response **before they can reach their logical destination**, the **required object can dynamically capture** the request or response
- Wrapper class
 - Creates the object to **capture the request and response** before they reach server and client respectively
 - The wrapper object generated by the filter **implements the `getWriter()` and `getOutputStream()`**, which returns a stand-in-stream. The stand-in-stream is passed to the servlet through the wrapper object
 - The wrapper object captures **the response through the stand-in-stream** and sends it back to the filter

Classes	Descriptions
ServletRequestWrapper	<ul style="list-style-type: none">- Provides a convenient implementation of the <code>ServletRequest</code> interface- Can be sub-classed by developers wishing to send the request to a servlet- To override request methods, one should wrap the request in an object that extends <code>ServletRequestWrapper</code> or <code>HttpServletRequestWrapper</code>
ServletResponseWrapper	<ul style="list-style-type: none">- Provides a convenient implementation of the <code>ServletResponse</code> interface- Can be sub classed by developers wishing to send the response from a servlet.

Filter Chain

Wrapper Class – Altering Request

- Create filter class extends to the **ServletRequestWrapper** or **HttpServletRequestWrapper** class.
- The object captures the **HttpRequest** object from the client and sends it to the filters
- Through the objects filter extends some services to the request.

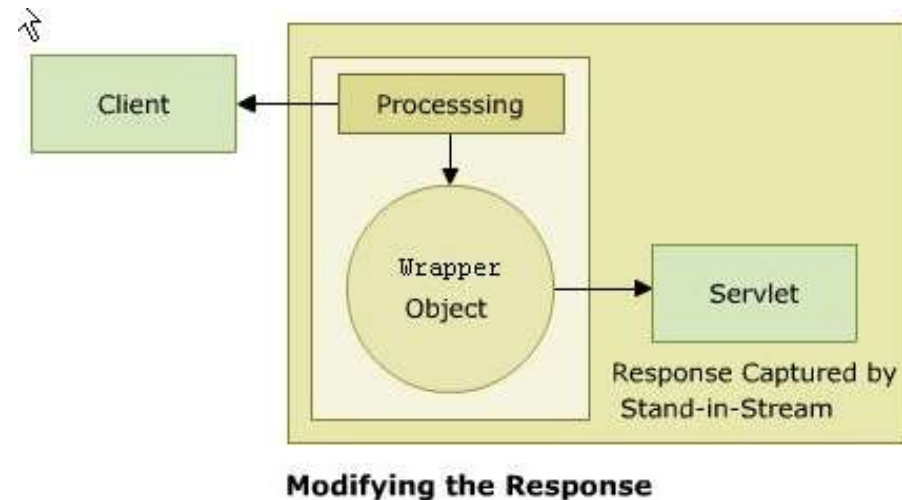


Modifying the Request

Filter Chain

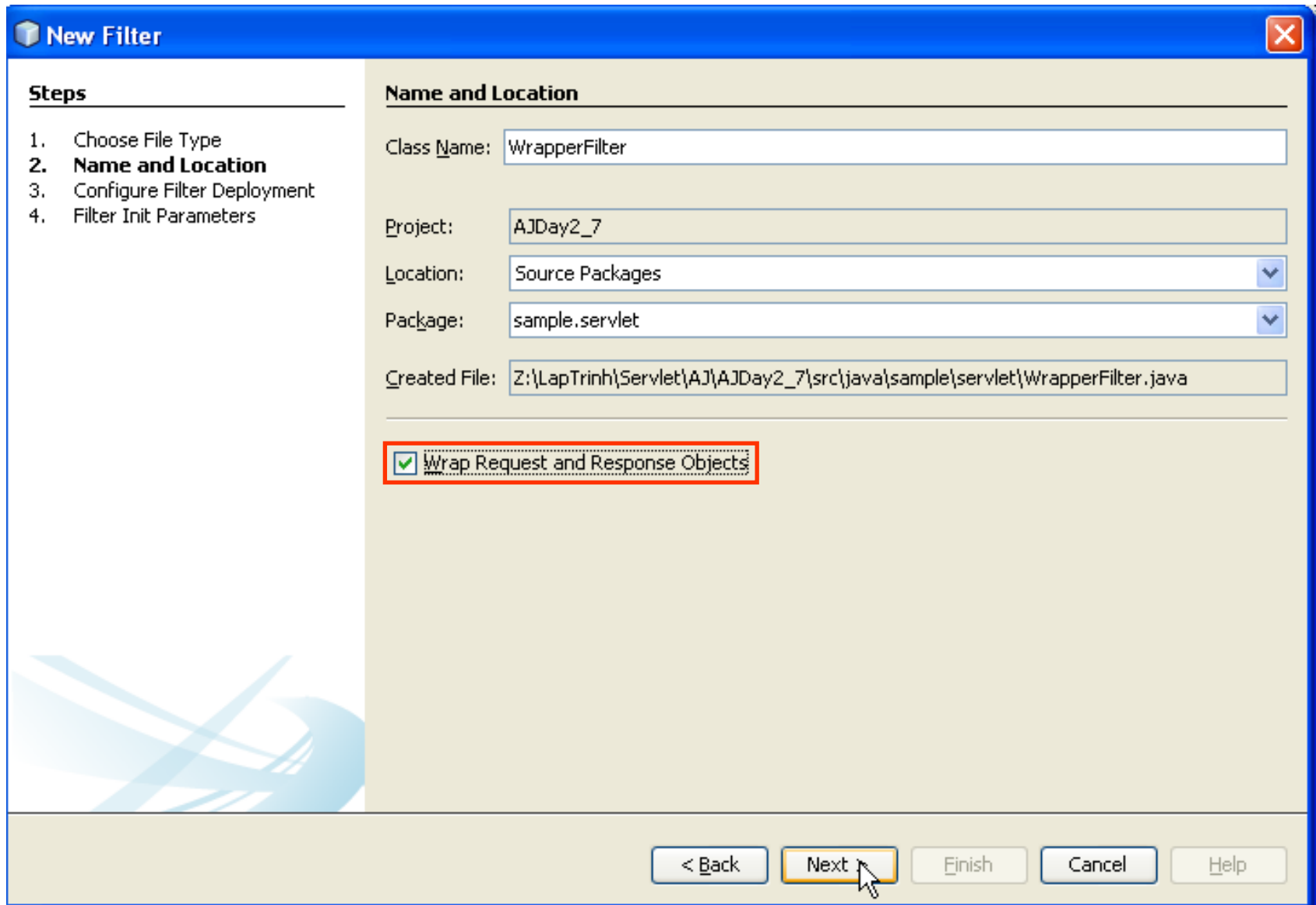
Wrapper Class – Altering Response

- Create filter class extends to the **ServletResponseWrapper** or **HttpServletResponseWrapper** class.
- The object captures the **HttpServletRequest** object from the client and sends it to the filters
- Through the objects filter extends some services to the request.



Filter Chain

Wrapper Class – Example



The image shows a 'New Filter' dialog box with a blue title bar and a close button. It contains a 'Steps' list on the left and a 'Name and Location' section on the right. The 'Steps' list has four items: '1. Choose File Type', '2. Name and Location' (which is bolded), '3. Configure Filter Deployment', and '4. Filter Init Parameters'. The 'Name and Location' section has several fields: 'Class Name' (WrapperFilter), 'Project' (AJDay2_7), 'Location' (Source Packages), 'Package' (sample.servlet), and 'Created File' (Z:\LapTrinh\Servlet\AJ\AJDay2_7\src\java\sample\servlet\WrapperFilter.java). A checkbox labeled 'Wrap Request and Response Objects' is checked and highlighted with a red rectangle. At the bottom, there are buttons for '< Back', 'Next >' (which is highlighted with a yellow border and a mouse cursor), 'Finish', 'Cancel', and 'Help'.

New Filter

Steps

1. Choose File Type
- 2. Name and Location**
3. Configure Filter Deployment
4. Filter Init Parameters

Name and Location

Class Name:

Project:

Location:

Package:



Created File:

☒ **Wrap Request and Response Objects**

< Back **Next >** Finish Cancel Help

Filter Chain

Wrapper Class – Example


New Filter


Steps

1. Choose File Type
2. Name and Location
3. **Configure Filter Deployment**
4. Filter Init Parameters

Configure Filter Deployment

Register the Filter with the application by giving the Filter an internal name. Describe when the Filter is invoked by listing the HTTP request path patterns or Servlets to which the Filter applies. Order this Filter's mappings relative to any other Filter invocation.

Class Name:

Filter Name:

Filter Mappings:

Filter name	Applies to
FilterWrapper	/FilterServlet
FirstFilter	/FilterServlet
SecondFilter	/FilterServlet

New...

Edit...

Delete

Move Up

Move Down

< Back

Next >

Finish

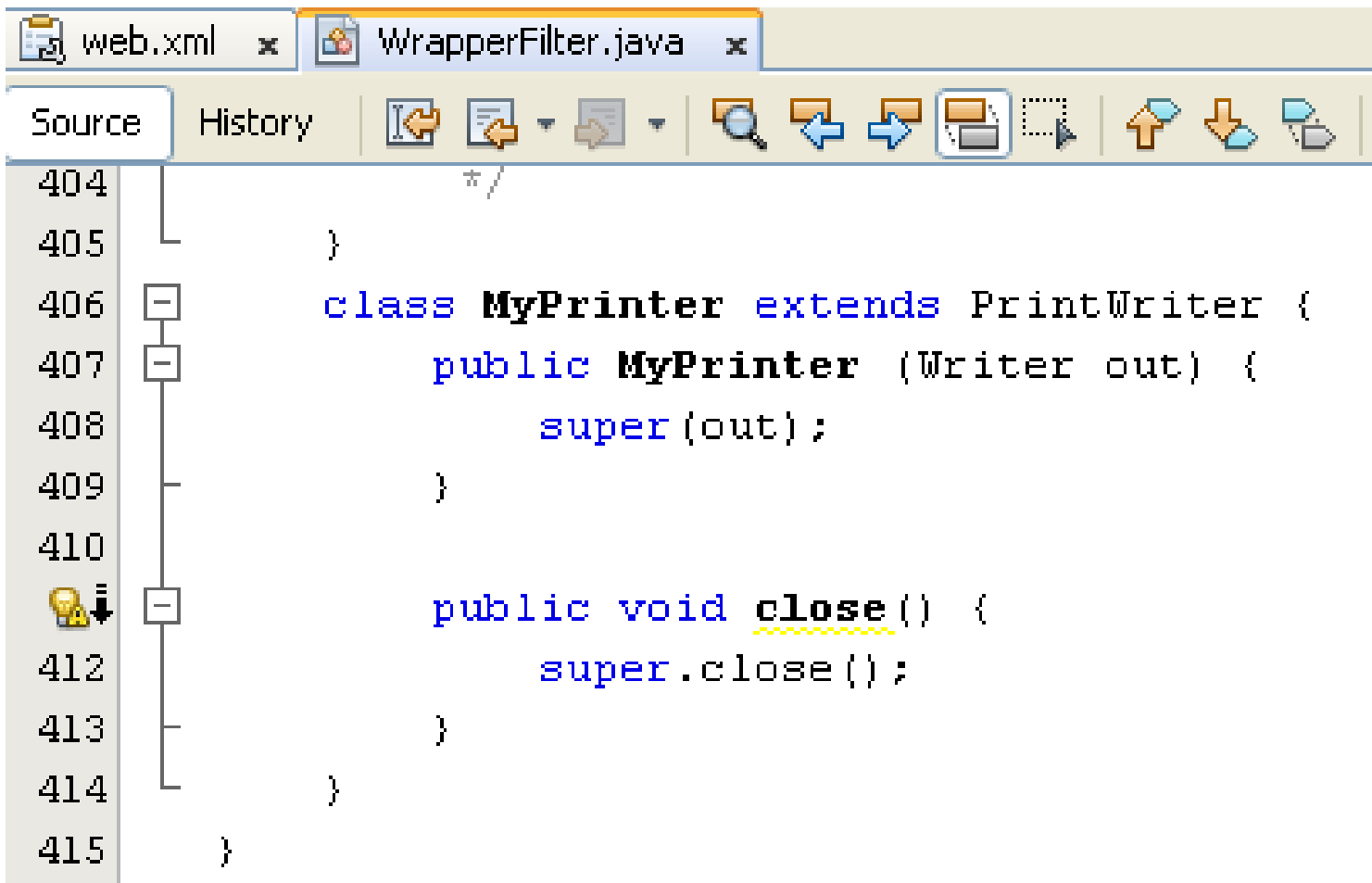
Cancel

Help

Filter Chain

Wrapper Class – Example

- Adding the MyPrinter class extends PrintWriter in FilterWrapper class



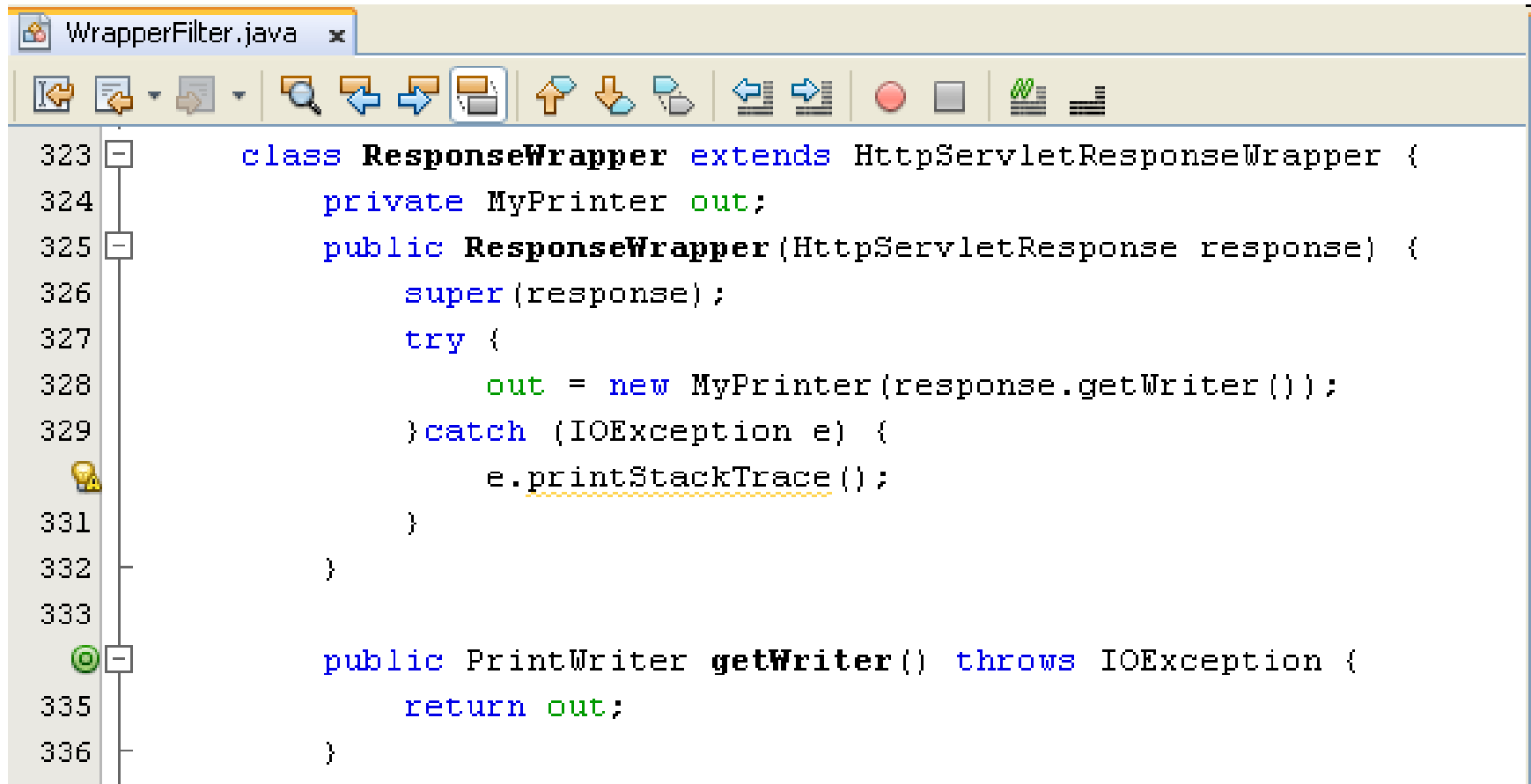
```

404  */
405  }
406  class MyPrinter extends PrintWriter {
407      public MyPrinter (Writer out) {
408          super(out);
409      }
410
411      public void close() {
412          super.close();
413      }
414  }
415  }
  
```

Filter Chain

Wrapper Class – Example

- Modifying the ResponseWrapper class uses MyPrinter to output stream



```

323 class ResponseWrapper extends HttpServletResponseWrapper {
324     private MyPrinter out;
325     public ResponseWrapper(HttpServletResponse response) {
326         super(response);
327         try {
328             out = new MyPrinter(response.getWriter());
329         } catch (IOException e) {
330             e.printStackTrace();
331         }
332     }
333
334     public PrintWriter getWriter() throws IOException {
335         return out;
336     }
  
```

Filter Chain

Wrapper Class – Example

```

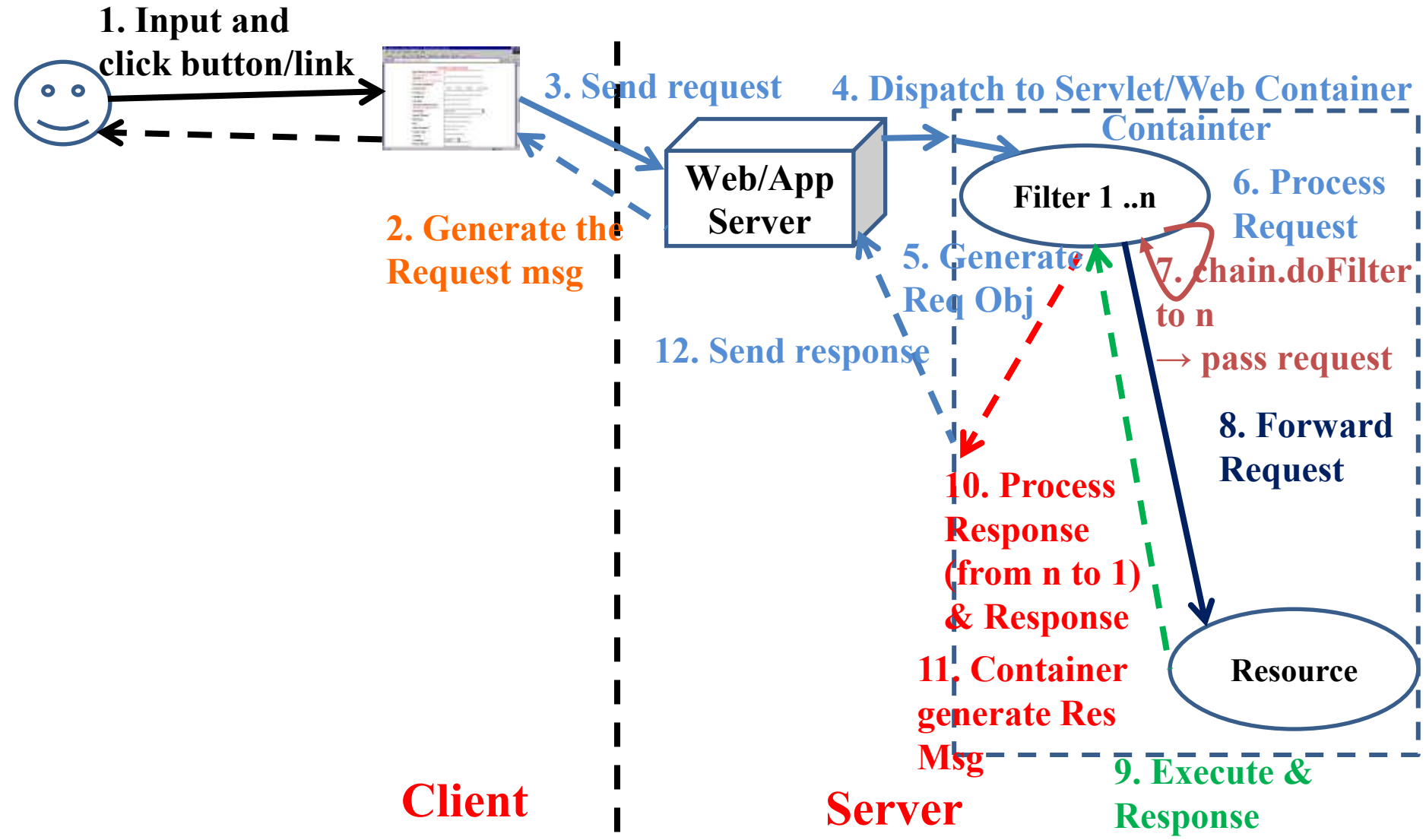
WrapperFilter.java x
133  */
134  public void doFilter(ServletRequest request, ServletResponse response,
135                      FilterChain chain)
136      throws IOException, ServletException {
137      HttpServletResponse resp = (HttpServletResponse)response;
138      ResponseWrapper wrapperResp = new ResponseWrapper(resp);
139      try {
140          chain.doFilter(request, wrapperResp);
141          PrintWriter out = wrapperResp.getWriter();
142          out.println("<br/>The slide is licensed to KhanhKT");
143          out.close();
144      }
145      catch(Throwable t) {
146          t.printStackTrace();
147      }
148  }
  
```

MVC2

Filter Acts as Controller

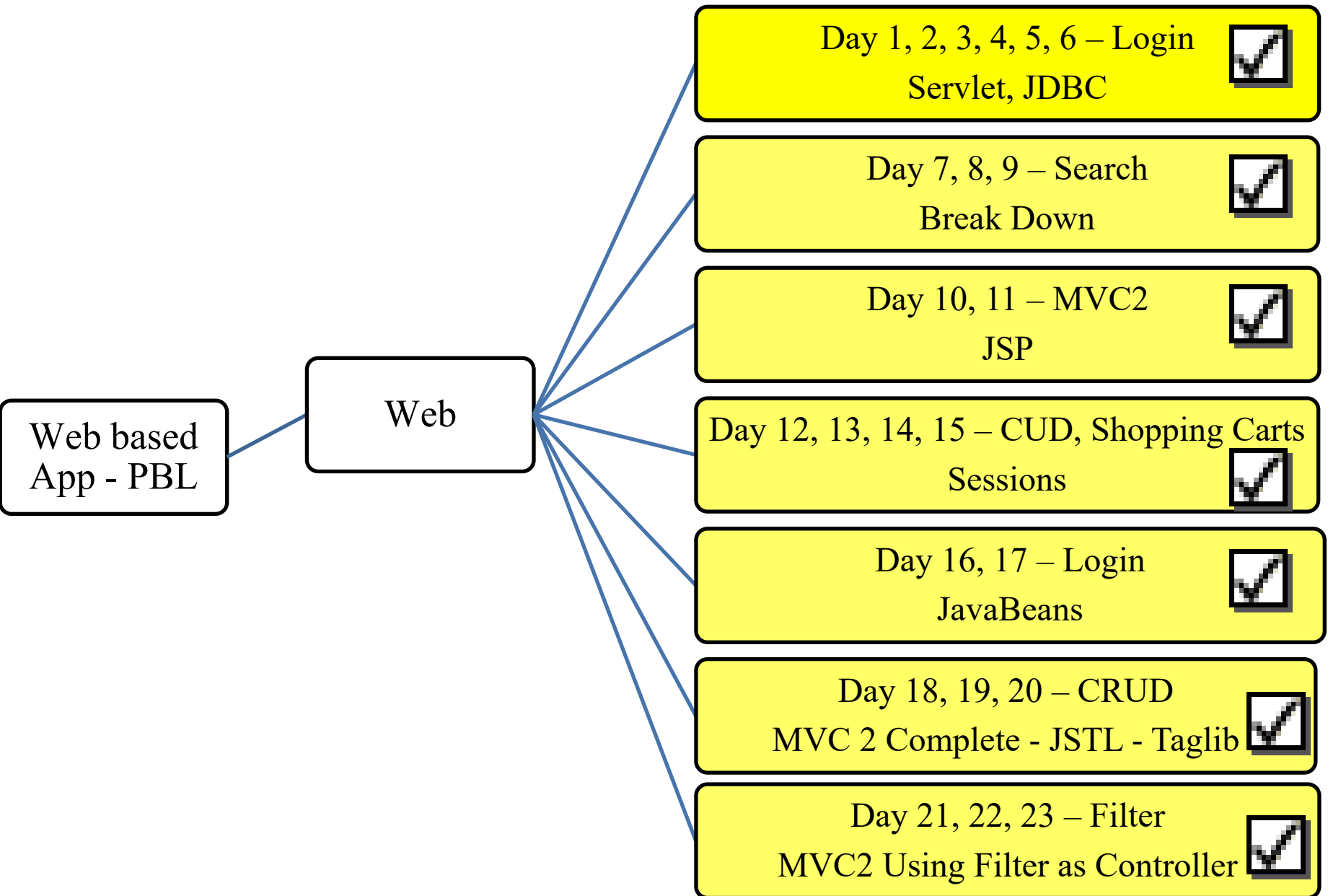
- While a **servlet** is the **most common controller**, a **filter** can **act as a controller** too
- While a **servlet** only **handles access** to the **dynamic** part of the application, a **filter** can **serve all** the **resources** in application, **including static ones**
- A filter as the controller **allows** to **block all requests** to the application, including request for static contents

Summary



Q&A

Summary



Next Days

- Practice with exercises
- Practical Test
- Presentations

Good luck to you!!!

Appendix

MVC2 using Filter as Controller

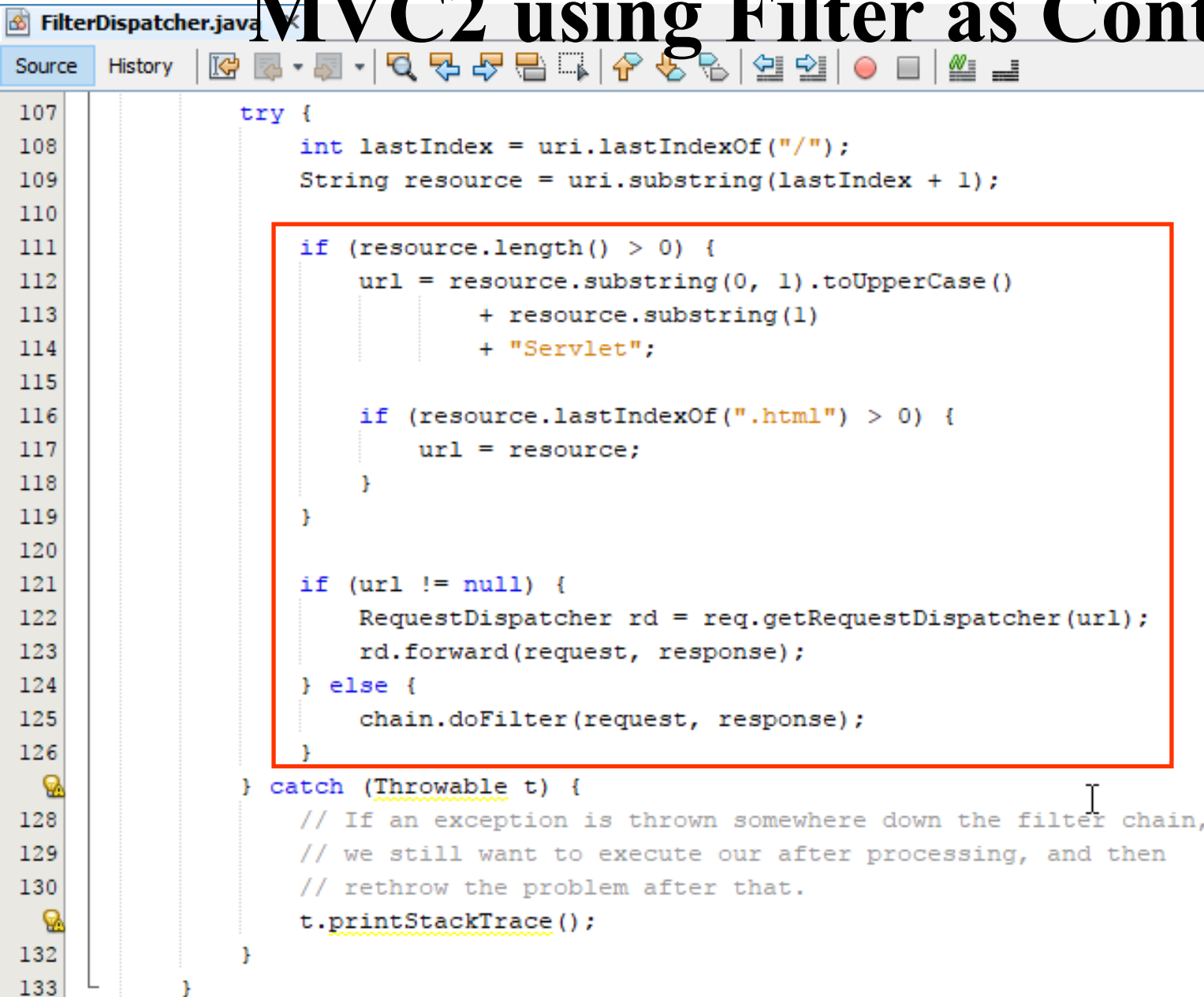
```

FilterDispatcher.java
Source History
23  * @author Kieu Trong Khanh
24  */
25  public class FilterDispatcher implements Filter {
26
27      private static final boolean debug = true;
28      private final String loginPage = "login.html";
29
30      // The filter configuration object we are associated with. If
31      // this value is null, this filter instance is not currently
32      // configured.
33      private FilterConfig filterConfig = null;
34
35      public FilterDispatcher() {...2 lines }
36
37
38      private void doBeforeProcessing(ServletRequest request, ServletResponse response)
39          throws IOException, ServletException {...26 lines }
40
41
42      private void doAfterProcessing(ServletRequest request, ServletResponse response)
43          throws IOException, ServletException {...23 lines }
44
45      /**...9 lines */
46      public void doFilter(ServletRequest request, ServletResponse response,
47                          FilterChain chain)
48          throws IOException, ServletException {
49          HttpServletRequest req = (HttpServletRequest) request;
50          String uri = req.getRequestURI();
51          String url = loginPage;
52
53      }
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105

```


Appendix

MVC2 using Filter as Controller

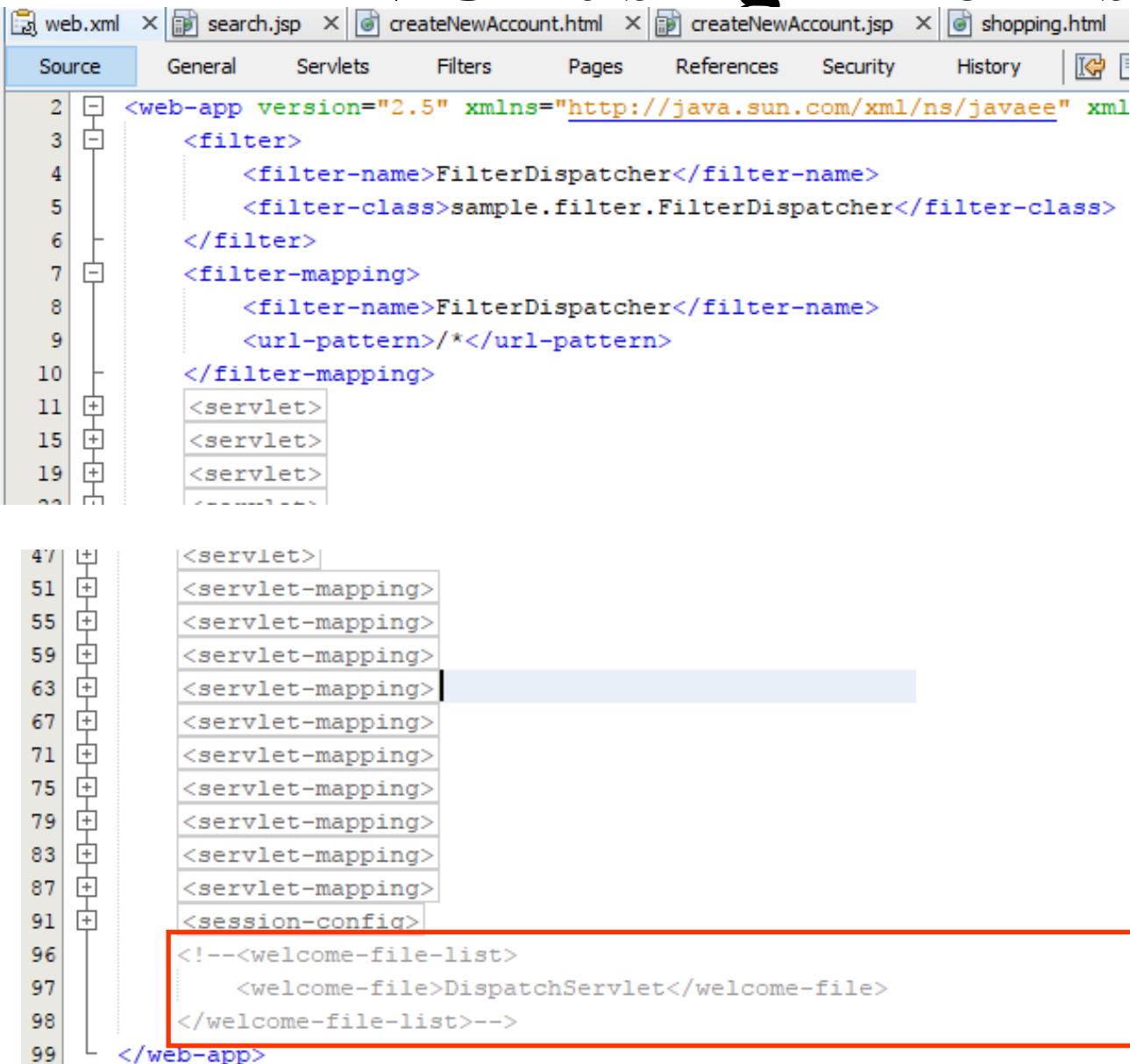


```

107     try {
108         int lastIndex = uri.lastIndexOf("/");
109         String resource = uri.substring(lastIndex + 1);
110
111         if (resource.length() > 0) {
112             url = resource.substring(0, 1).toUpperCase()
113                 + resource.substring(1)
114                 + "Servlet";
115
116             if (resource.lastIndexOf(".html") > 0) {
117                 url = resource;
118             }
119         }
120
121         if (url != null) {
122             RequestDispatcher rd = req.getRequestDispatcher(url);
123             rd.forward(request, response);
124         } else {
125             chain.doFilter(request, response);
126         }
127     } catch (Throwable t) {
128         // If an exception is thrown somewhere down the filter chain,
129         // we still want to execute our after processing, and then
130         // rethrow the problem after that.
131         t.printStackTrace();
132     }
133 }
  
```

Appendix

MVC2 using Filter as Controller



The screenshot shows an IDE with several tabs open: web.xml, search.jsp, createNewAccount.html, createNewAccount.jsp, and shopping.html. The 'Source' tab is active, displaying the contents of web.xml. The code is as follows:

```

2  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xml
3
4  <filter>
5      <filter-name>FilterDispatcher</filter-name>
6      <filter-class>sample.filter.FilterDispatcher</filter-class>
7  </filter>
8  <filter-mapping>
9      <filter-name>FilterDispatcher</filter-name>
10     <url-pattern>/*</url-pattern>
11 </filter-mapping>
12 <servlet>
13
14 <servlet>
15
16 <servlet>
17
18 <servlet>
19
20 <servlet>
21
22 <servlet>
23
24 <servlet>
25
26 <servlet>
27
28 <servlet>
29
30 <servlet>
31
32 <servlet>
33
34 <servlet>
35
36 <servlet>
37
38 <servlet>
39
40 <servlet>
41
42 <servlet>
43
44 <servlet>
45
46 <servlet>
47
48 <servlet>
49
50 <servlet>
51
52 <servlet>
53
54 <servlet>
55
56 <servlet>
57
58 <servlet>
59
60 <servlet>
61
62 <servlet>
63
64 <servlet>
65
66 <servlet>
67
68 <servlet>
69
70 <servlet>
71
72 <servlet>
73
74 <servlet>
75
76 <servlet>
77
78 <servlet>
79
80 <servlet>
81
82 <servlet>
83
84 <servlet>
85
86 <servlet>
87
88 <servlet>
89
90 <servlet>
91
92 <servlet>
93
94 <servlet>
95
96 <!--<welcome-file-list>
97     <welcome-file>DispatchServlet</welcome-file>
98 </welcome-file-list>-->
99 </web-app>

```

The code defines a FilterDispatcher filter that applies to all URL patterns. It also defines several servlets and mappings. A red box highlights the welcome-file-list section, which is commented out.

Appendix

MVC2 using Filter as Controller

```

login.html x search.jsp x createNewAccount.html x createNewAccount.jsp x shopping.html x viewCart
Source History
1 <!DOCTYPE html>
2 ...5 lines
7 <html>
8   <head>
9     <title>Login</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-width, initial-scale=1.0">
12  </head>
13  <body>
14    <h1>Login Page</h1>
15    <form action="login" method="POST">
16      Username <input type="text" name="txtUsername" value="" /><br/>
17      Password <input type="password" name="txtPassword" value="" /><br/>
18      <input type="submit" value="Login" name="btAction" />
19      <input type="reset" value="Reset" />
20    </form><br/>
21    <a href="shopping.html">Click here to buy books</a><br/>
22    <a href="createNewAccount.html">Click here to Sign Up</a>
23  </body>
24 </html>

```

Appendix

MVC2 using Filter as Controller

```

search.jsp x createNewAccount.html x createNewAccount.jsp x shopping.html x viewCart.jsp x DeleteRecord
Source History
4      Author      : kieuokhanh
5      --%>
6
7      <%@page import="sample.registration.RegistrationDTO"%>
8      <%@page import="java.util.List"%>
9      <%@page contentType="text/html" pageEncoding="UTF-8"%>
10     <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
11     <!DOCTYPE html>
12     <html>
13     <head>
14         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15         <title>Search</title>
16     </head>
17     <body>
18         <font color="red">
19             Welcome, ${sessionScope.USERNAME}
20         </font>
21         <h1>Search Page</h1>
22         <form action="search">
23             Search Value <input type="text" name="txtSearchValue" value="" /><br/>
24             <input type="submit" value="Search" name="btAction" />
25         </form>
26         <br/>

```

Appendix

MVC2 using Filter as Controller

search.jsp × createNewAccount.html × createNewAccount.jsp × shopping.html × viewCart.jsp × DeleteRecordServlet.java × UpdateRecords

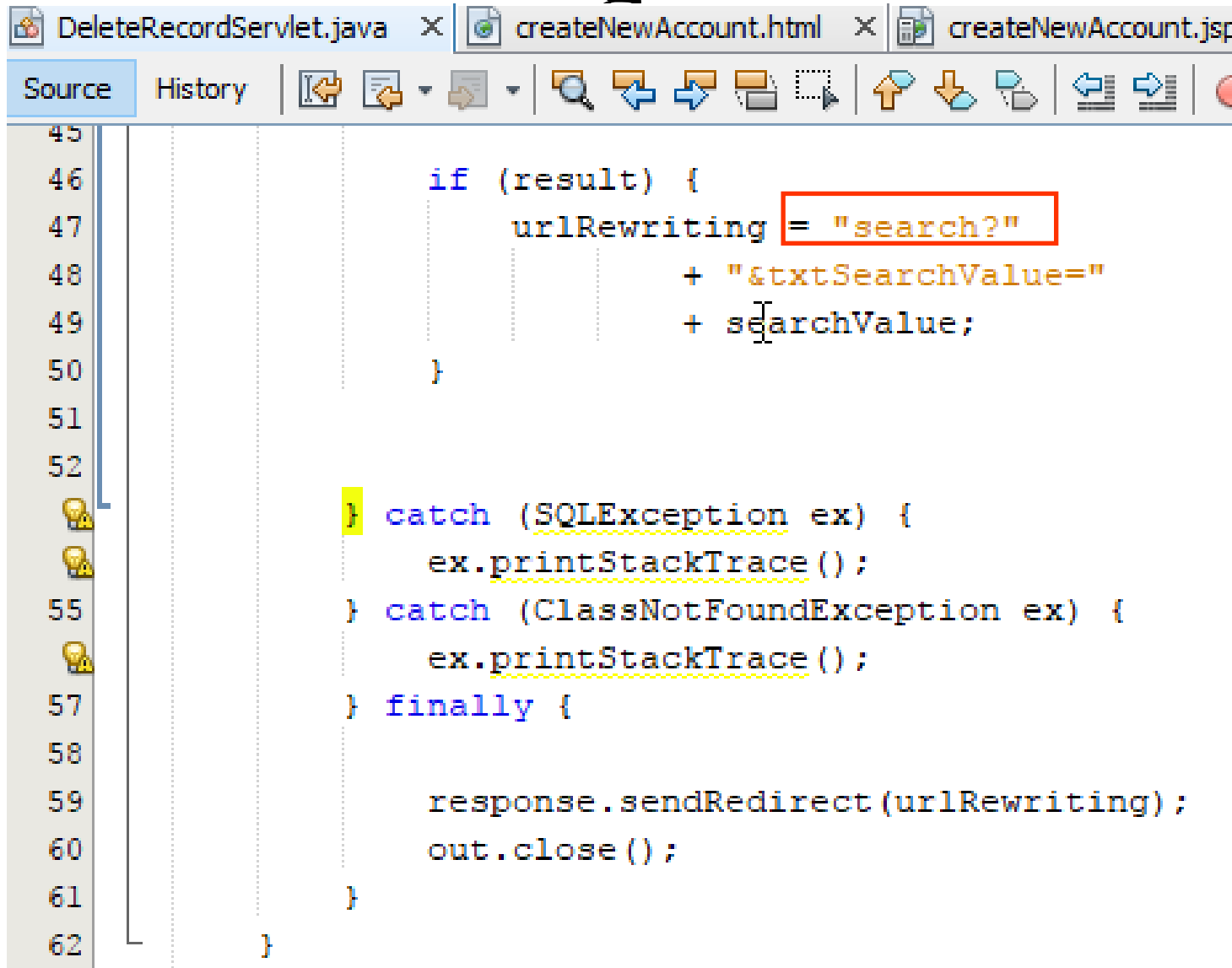
Source History

```

44 <tbody>
45 <c:forEach var="dto" items="${result}" varStatus="counter">
46 <form action="updateRecord">
47 <tr>
48 <...3 lines />
51 <...5 lines />
56 <...4 lines />
60 <...3 lines />
63 <...7 lines />
70 <td>
71 <c:url var="urlRewriting" value="deleteRecord">
72 <c:param name="pk" value="${dto.username}"/>
73 <c:param name="lastSearchValue" value="${param.txtSearchValue}"/>
74 </c:url>
75 <a href="${urlRewriting}">Delete</a>
76 </td>
77 <...5 lines />
82 </tr>
83 </form>
84 </c:forEach>
85 </tbody>
86 </table>
  
```

Appendix

MVC2 using Filter as Controller



```

45
46         if (result) {
47             urlRewriting = "search?"
48                             + "&txtSearchValue="
49                             + searchValue;
50         }
51
52
53     } catch (SQLException ex) {
54         ex.printStackTrace();
55     } catch (ClassNotFoundException ex) {
56         ex.printStackTrace();
57     } finally {
58
59         response.sendRedirect(urlRewriting);
60         out.close();
61     }
62 }
  
```

Appendix

MVC2 using Filter as Controller

```

UpdateRecordServlet.java x createNewAccount.html x createNewAccount.jsp x shopping.html x viewCart.
Source History
47 String urlRewriting = updateErrPage;
48 try {
49     RegistrationDAO dao = new RegistrationDAO();
50     boolean result = dao.updatePassRole(username, password, role);
51
52     if (result) {
53         urlRewriting = "search?"
54             + "&txtSearchValue="
55             + searchValue;
56     }
57 } catch (SQLException ex) {
58     ex.printStackTrace();
59 } catch (ClassNotFoundException ex) {
60     ex.printStackTrace();
61 } finally {
62     response.sendRedirect(urlRewriting);
63     out.close();
64 }
65 }

```

Appendix

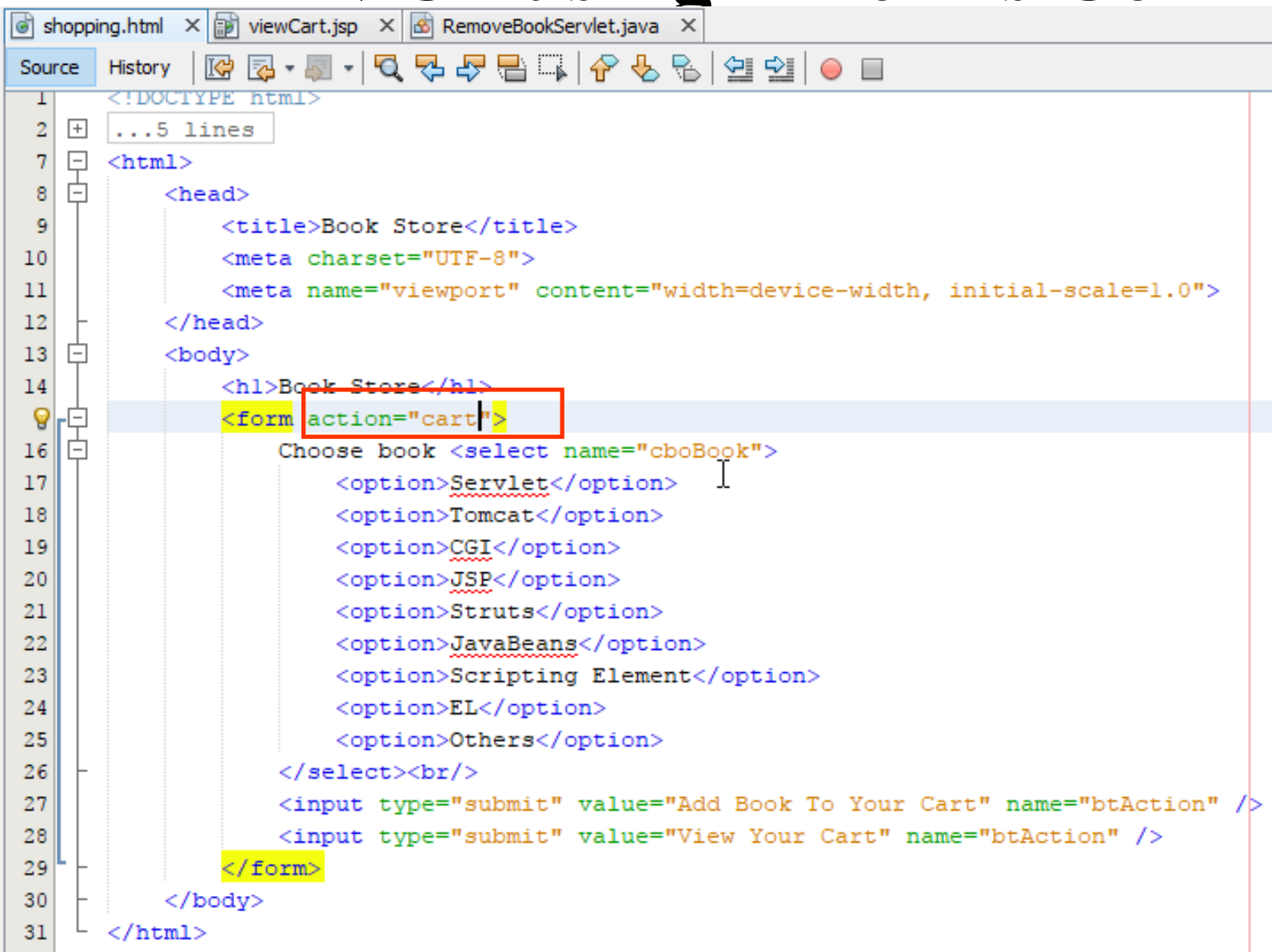
MVC2 using Filter as Controller

```

createNewAccount.html x createNewAccount.jsp x shopping.html x viewCart.jsp x RemoveBookServlet.java x
Source History
1 <!DOCTYPE html>
2 ...5 lines
7 <html>
8   <head>
9     <title>Create</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-width, initial-scale=1.0">
12  </head>
13  <body>
14    <h1>Create New Account</h1>
15    <form action="createRecord" method="POST">
16      Username* <input type="text" name="txtUsername" value="" />(6 - 20 chars)<br/>
17      Password* <input type="password" name="txtPassword" value="" />(6 - 30 chars)<br/>
18      Confirm <input type="password" name="txtConfirm" value="" /><br/>
19      Full name <input type="text" name="txtFullname" value="" />(2 - 50 chars)<br/>
20      <input type="submit" value="Create New Account" name="btAction" />
21      <input type="reset" value="Reset" />
22    </form>
23  </body>
24 </html>
  
```


Appendix

MVC2 using Filter as Controller



```

1 <!DOCTYPE html>
2 ...5 lines
7 <html>
8   <head>
9     <title>Book Store</title>
10    <meta charset="UTF-8">
11    <meta name="viewport" content="width=device-width, initial-scale=1.0">
12  </head>
13  <body>
14    <h1>Book Store</h1>
15    <form action="cart">
16      Choose book <select name="cboBook">
17        <option>Servlet</option>
18        <option>Tomcat</option>
19        <option>CGI</option>
20        <option>JSP</option>
21        <option>Struts</option>
22        <option>JavaBeans</option>
23        <option>Scripting Element</option>
24        <option>EL</option>
25        <option>Others</option>
26      </select><br/>
27      <input type="submit" value="Add Book To Your Cart" name="btAction" />
28      <input type="submit" value="View Your Cart" name="btAction" />
29    </form>
30  </body>
31 </html>
  
```

Appendix

MVC2 using Filter as Controller

CartServlet.java

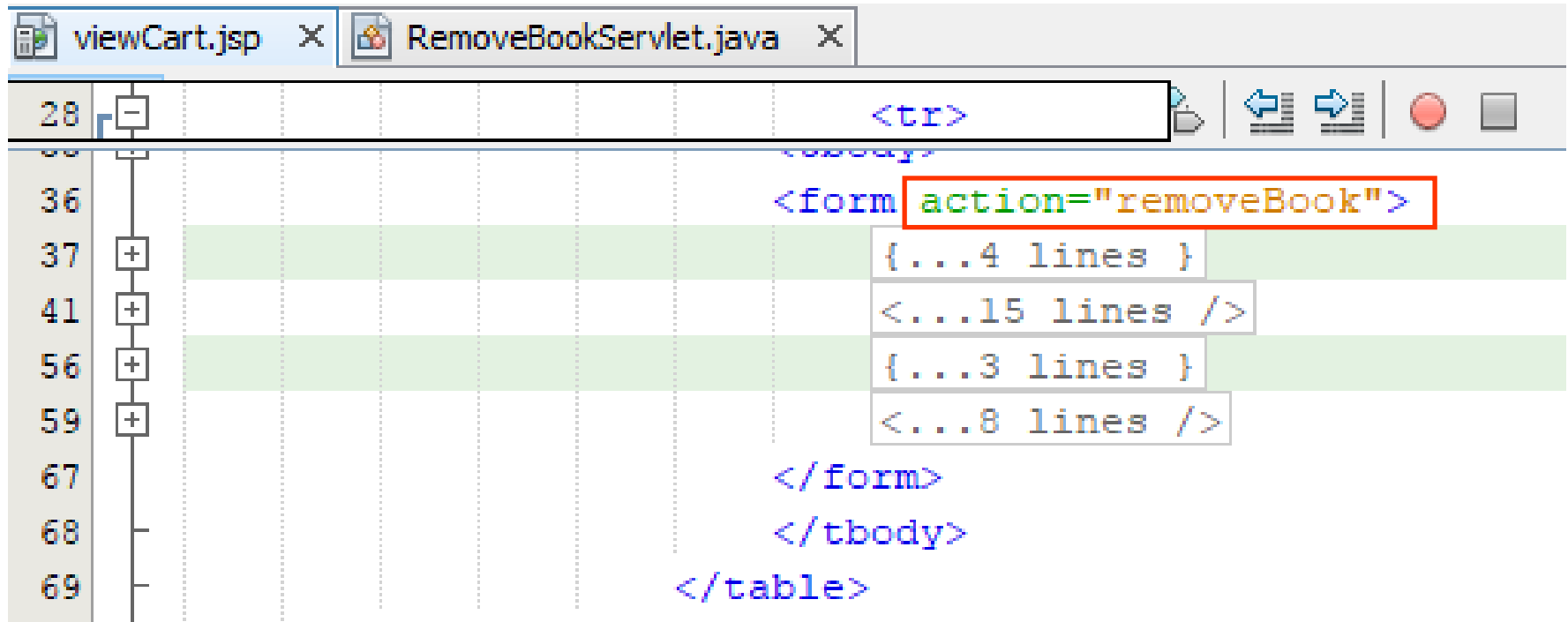
Source History

```

18  * @author Kieu Trong Khanh
19  */
20  public class CartServlet extends HttpServlet {
21
22      private final String addBookServlet = "AddBookServlet";
23      private final String viewCartPage = "viewCart.jsp";
24
25      /** Processes requests for both HTTP <code>GET</code> and <code>POST</code>
34  protected void processRequest(HttpServletRequest request, HttpServletResponse
35      throws ServletException, IOException {
36      response.setContentType("text/html;charset=UTF-8");
37      PrintWriter out = response.getWriter();
38
39      String action = request.getParameter("btAction");
40      String url = "";
41
42      try {
43          if (action.equals("Add Book To Your Cart")) {
44              url = addBookServlet;
45          } else if (action.equals("View Your Cart")) {
46              url = viewCartPage;
47          }
48      } finally {
49          RequestDispatcher rd = request.getRequestDispatcher(url);
50          rd.forward(request, response);
51          out.close();
52      }
53  }
  
```

Appendix

MVC2 using Filter as Controller



```


28  <tr>
35  </tbody>
36  <form action="removeBook">
37  { ... 4 lines }
41  <... 15 lines />
56  { ... 3 lines }
59  <... 8 lines />
67  </form>
68  </tbody>
69  </table>
  
```

Appendix

MVC2 using Filter as Controller

RemoveBookServlet.java

Source History



```

31
32     protected void processRequest(HttpServletRequest request, HttpServletResponse response)
33         throws ServletException, IOException {
34         response.setContentType("text/html;charset=UTF-8");
35         PrintWriter out = response.getWriter();
36
37         String urlRewriting = "cart?btAction=View Your Cart";
38
39         try {
40             HttpSession session = request.getSession(false);
41
42             if (session != null) {
43                 CartObj cart = (CartObj)session.getAttribute("CART");
44
45                 if (cart != null) {

```

Appendix

MVC2 using Filter as Controller

Projects

