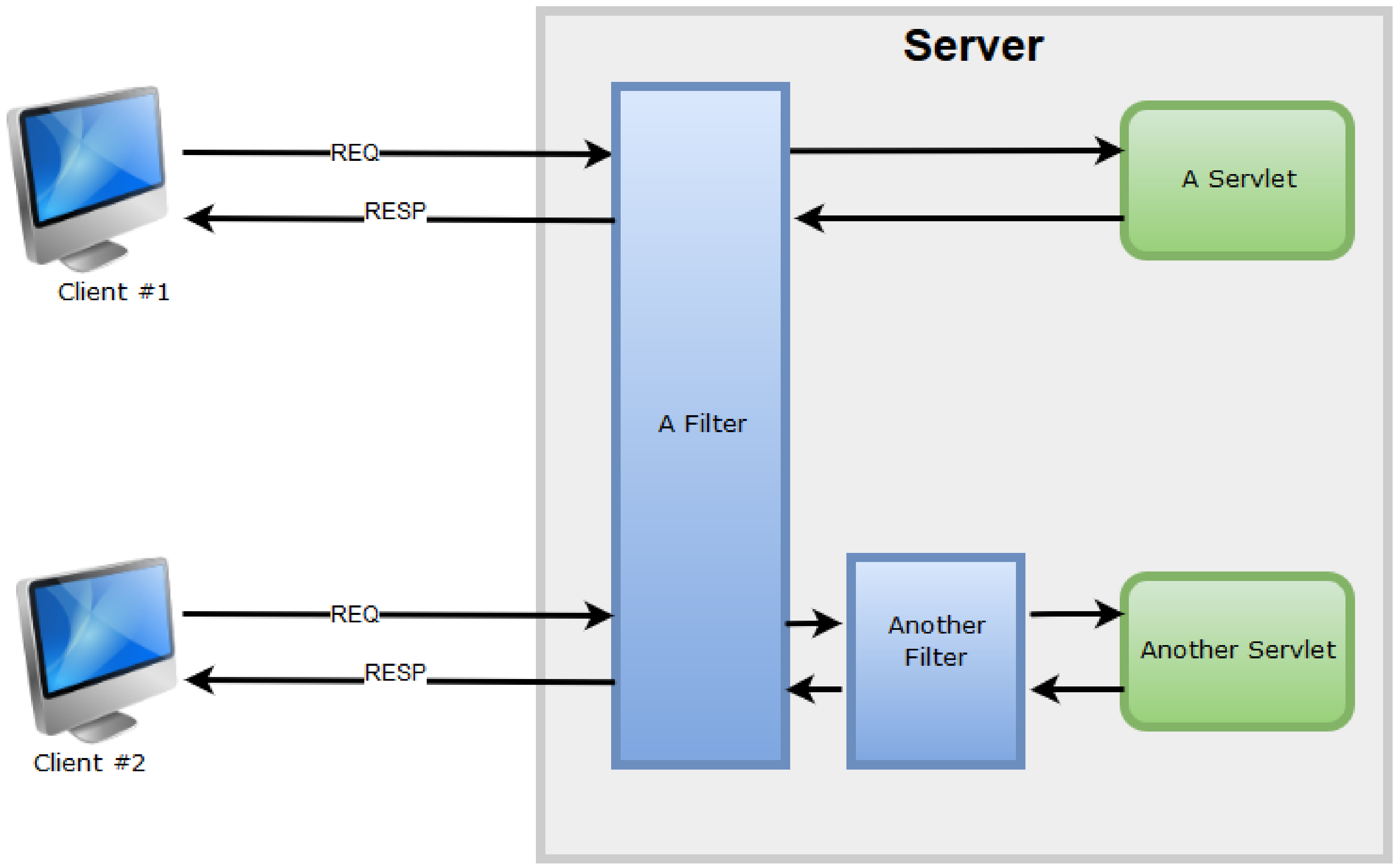
B.Sc. In Software Development. Year 4.
Distributed Object Based Systems.
Using Filters

LIMERICK INSTITUTE OF TECHNOLOGY
SCHOOL OF SCIENCE, ENGINEERING & I.T.
Department of Information Technology

Introduction

- Introduced to the Servlet specification with version 2.3.
- You can add a filter to your web application that intercepts a request and then executes some code before or after a Servlet (or JSP) is executed.
- Sometimes this code may modify the response that’s sent back to clients.
- Filters are ideal for cross cutting concerns.

Introduction



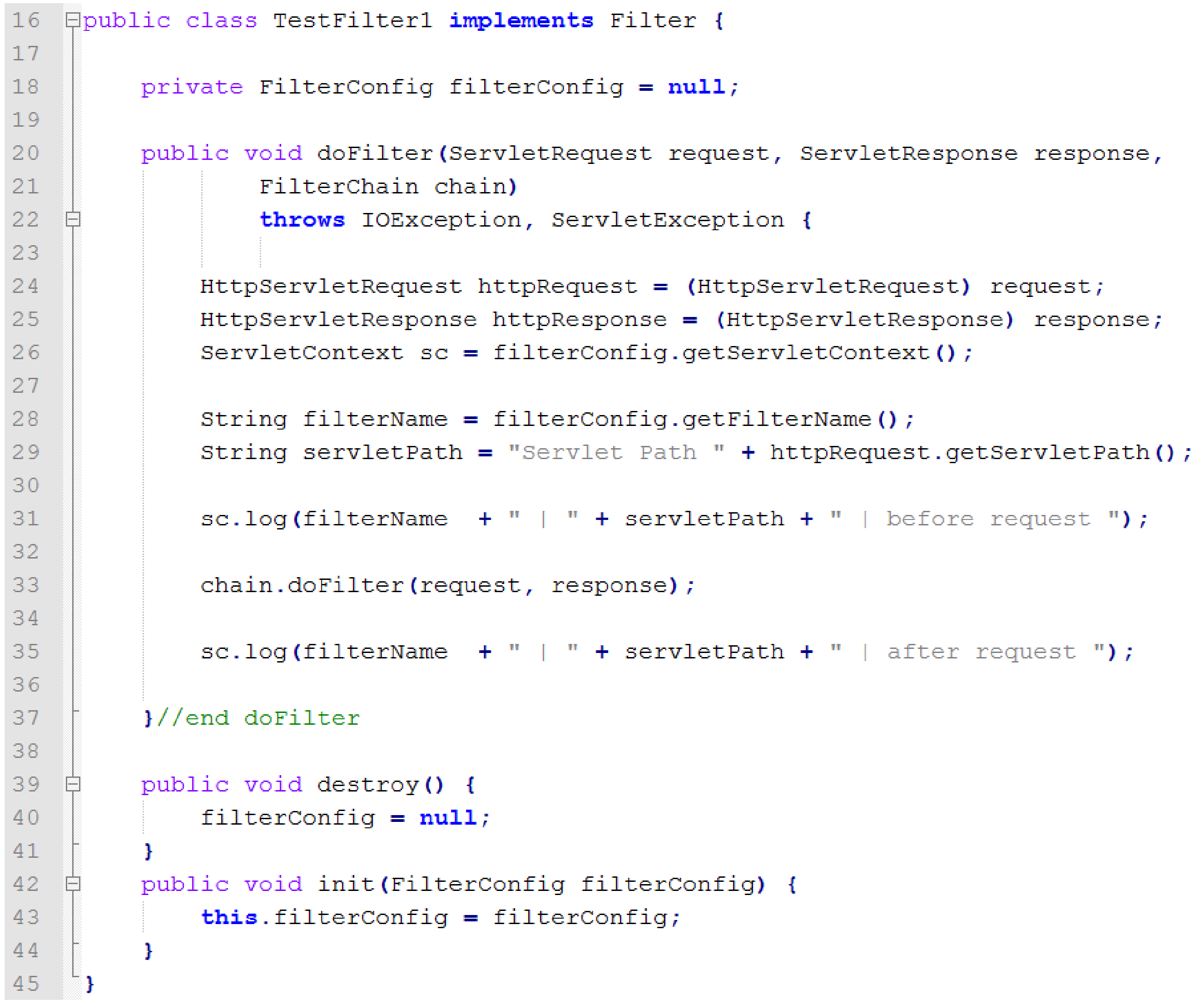
Benefits and Uses

- One benefit of filters is that they allow you to create **modular code** that can be applied to different parts of an application.
- Another benefit of filters, is that they allow you to create **flexible code**.
    - Use an applications web.xml file to control when filters are executed
- A filter can be used to write data to a log file, handle authentication or compress a response.
- You could use a filter to vary the processing based on the data that’s in the request.

How to add a filter

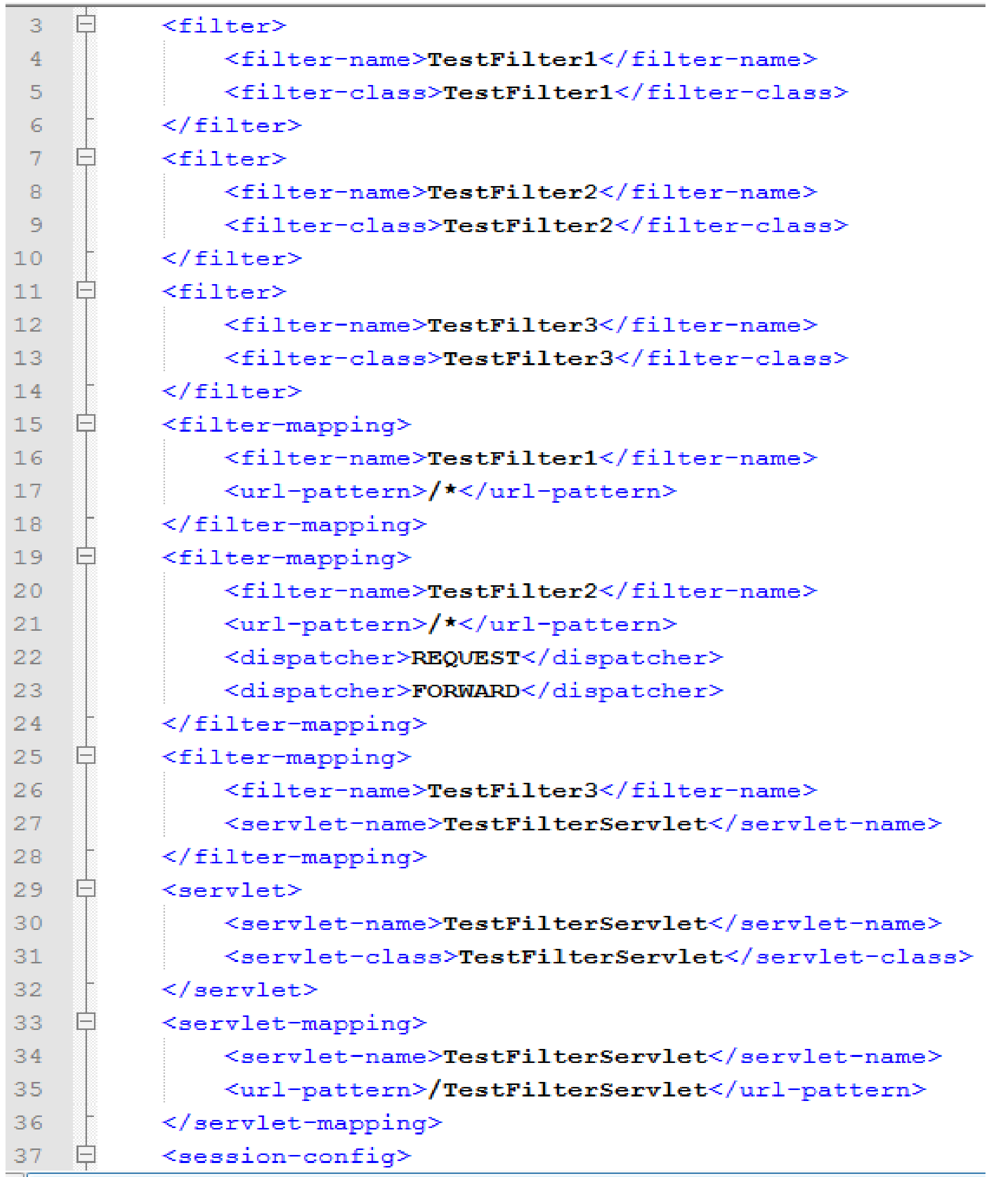
- To start you must code a class for the filter, then add some code to the web.xml file to map the filter to one or more URL patterns.
- The code for our first filter appears on the next slide.
  - As simple as this example is, it illustrates all of the principles that you need for coding a filter.
  - It just essentially writes some information to a log.

How to add a filter



```java
public class TestFilter1 implements Filter {

    private FilterConfig filterConfig = null;

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

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;
        ServletContext sc = filterConfig.getServletContext();

        String filterName = filterConfig.getFilterName();
        String servletPath = "Servlet Path " + httpRequest.getServletPath();

        sc.log(filterName  + " | " + servletPath + " | before request ");

        chain.doFilter(request, response);

        sc.log(filterName  + " | " + servletPath + " | after request ");

    }//end doFilter

    public void destroy() {
        filterConfig = null;
    }
    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }
}
```

How to configure a filter



```xml
    <filter>
        <filter-name>TestFilter1</filter-name>
        <filter-class>TestFilter1</filter-class>
    </filter>
    <filter>
        <filter-name>TestFilter2</filter-name>
        <filter-class>TestFilter2</filter-class>
    </filter>
    <filter>
        <filter-name>TestFilter3</filter-name>
        <filter-class>TestFilter3</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>TestFilter1</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    <filter-mapping>
        <filter-name>TestFilter2</filter-name>
        <url-pattern>/*</url-pattern>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
    </filter-mapping>
    <filter-mapping>
        <filter-name>TestFilter3</filter-name>
        <servlet-name>TestFilterServlet</servlet-name>
    </filter-mapping>
    <servlet>
        <servlet-name>TestFilterServlet</servlet-name>
        <servlet-class>TestFilterServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>TestFilterServlet</servlet-name>
        <url-pattern>/TestFilterServlet</url-pattern>
    </servlet-mapping>
    <session-config>
```

Partial listing of web.xml

How to configure a filter

- The listing on the previous slide configures three filters.
- Except for the name of the class, all three filters contain the same code as the TestFilter1.
- The three filters are mapped to a URL pattern.
- The first filter mapping element maps TestFilter1 to all URL requests within the current element.
  - To do that, the url-pattern element uses a front slash followed by an asterisk.
  - As a result, this filter is executed for all URL’s in the root directory.

How to configure a filter

- The second filter mapping also maps TestFilter2 to all URL requests within the current application.
- However, this element includes two dispatcher elements that indicate that this filter should be executed for (1) requests coming from clients and (2) requests that are forwarded from within the application.
- By contrast, TestFilter1 is only executed for requests coming from clients.

How to configure a filter

- The third filter mapping uses the servlet-name element to map TestFilter3 to all requests for the TestFilterServlet.
- Working with this xml file is pretty easy.
- For example, you can easily turn off TestFilter1 by commenting out its servlet-mapping element.
- Alternatively, you can change the URL’s that cause TestFilter2 to be executed by modifying its url-pattern element.
- Once you do that, you don’t have to compile or modify your filter or servlet classes.

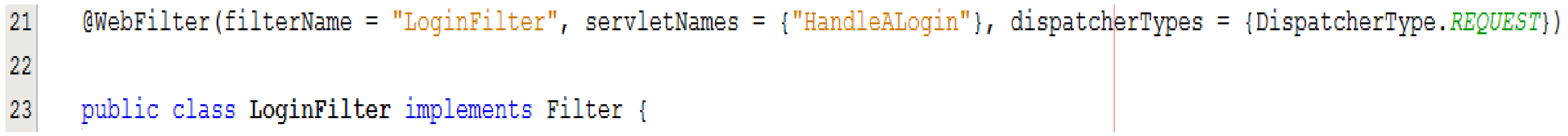
How to configure a filter

Filter Mapping Elements

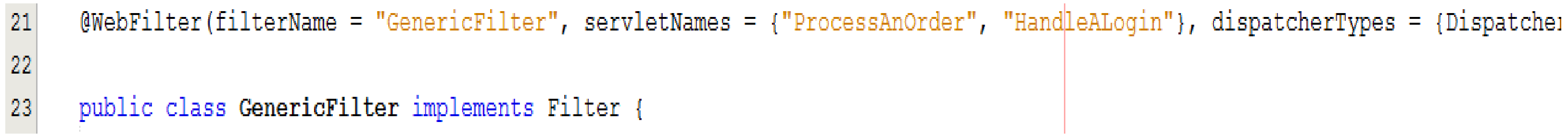
Element	Description
filter	Add a filter to the application
filter-name	The name of the filter
filter-class	The name of the class than implements the filter
filter-mapping	The filter mapping for the application
url-pattern	The URL(s) that result in the filter being called
servlet-name	The Servlet that results in the filter being called.
dispatcher	The types of requests that result in the filter being called. Values include REQUEST (default), FORWARD, ERROR and INCLUDE.

How to configure a filter

- Since V3.0 of the Servlet spec annotations can be used to declare filters.
- This annotation to define init parameters, filter name and description, servlets, url patterns and dispatcher types to apply the filter
- If you make frequent changes to the filter configurations, its better to use web.xml.



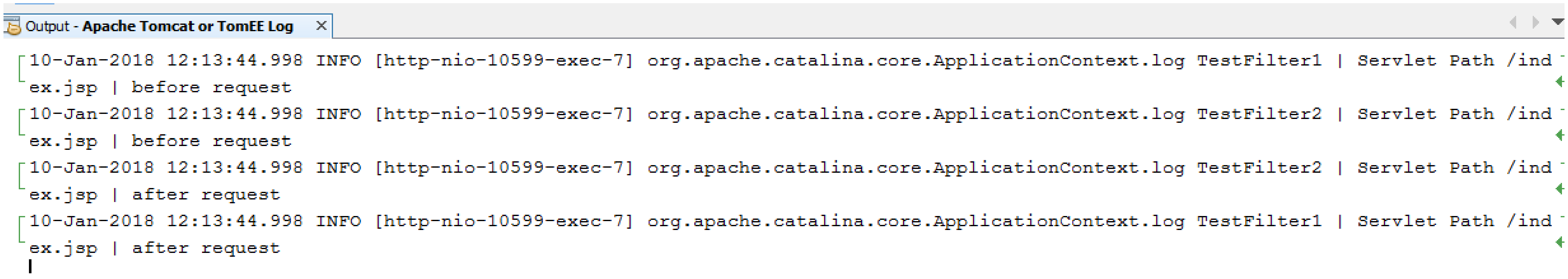
```java
@WebFilter(filterName = "LoginFilter", servletNames = {"HandleALogin"}, dispatcherTypes = {DispatcherType.REQUEST})

public class LoginFilter implements Filter {
```



```java
@WebFilter(filterName = "GenericFilter", servletNames = {"ProcessAnOrder", "HandleALogin"}, dispatcherTypes = {Dispatche

public class GenericFilter implements Filter {
```

Output from the example

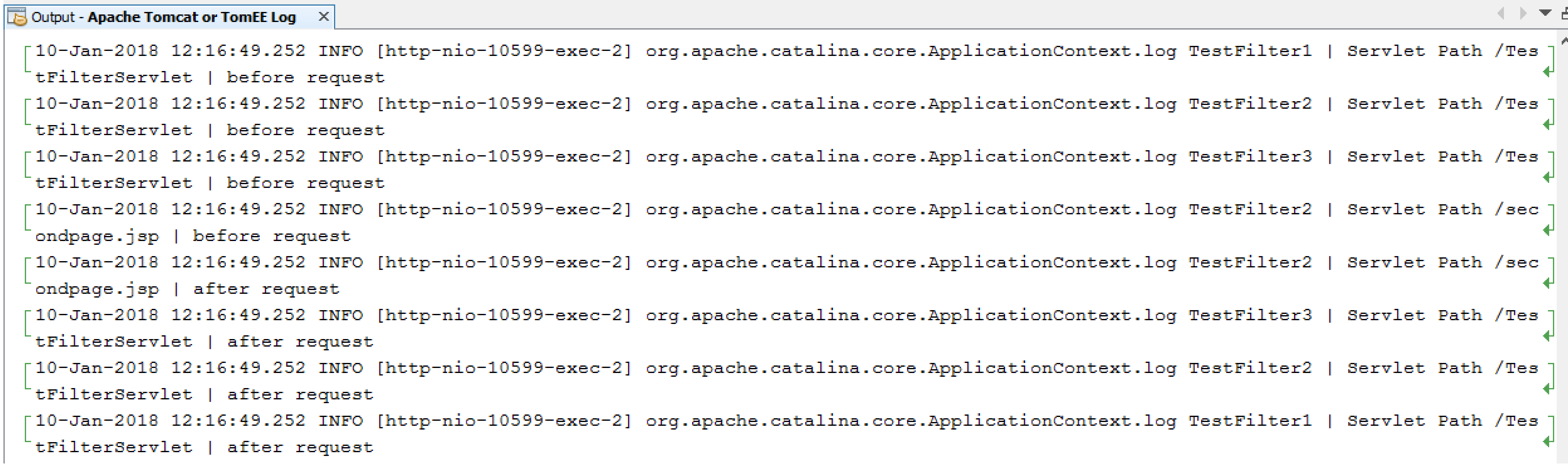
- Output when index.jsp is requested:



```
10-Jan-2018 12:13:44.998 INFO [http-nio-10599-exec-7] org.apache.catalina.core.ApplicationContext.log TestFilter1 | Servlet Path /ind
ex.jsp | before request
10-Jan-2018 12:13:44.998 INFO [http-nio-10599-exec-7] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /ind
ex.jsp | before request
10-Jan-2018 12:13:44.998 INFO [http-nio-10599-exec-7] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /ind
ex.jsp | after request
10-Jan-2018 12:13:44.998 INFO [http-nio-10599-exec-7] org.apache.catalina.core.ApplicationContext.log TestFilter1 | Servlet Path /ind
ex.jsp | after request
```

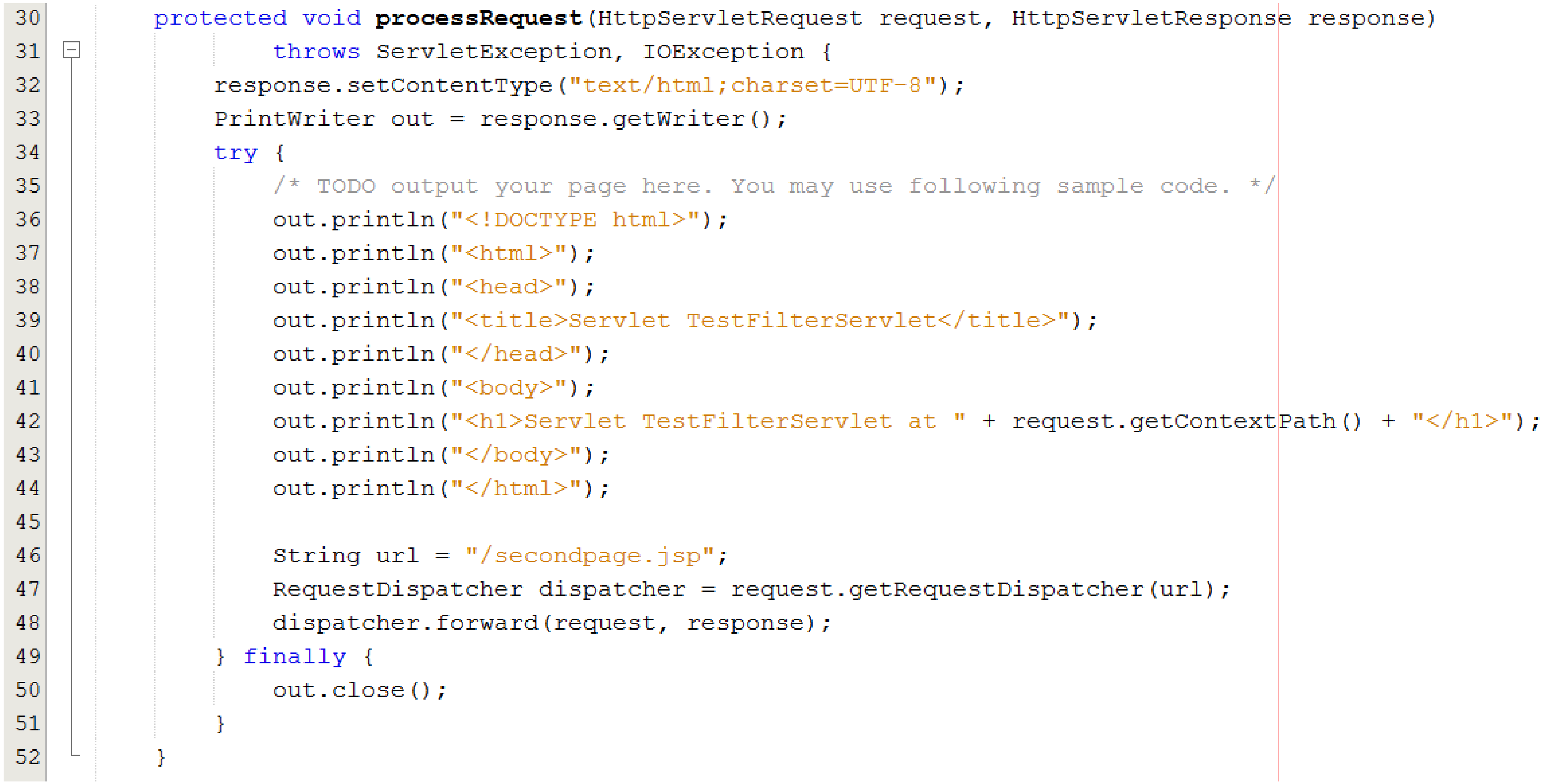
Output from the example

- Output when TestFilterServlet is requested:



```
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter1 | Servlet Path /Tes
tFilterServlet | before request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /Tes
tFilterServlet | before request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter3 | Servlet Path /Tes
tFilterServlet | before request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /sec
ondpage.jsp | before request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /sec
ondpage.jsp | after request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter3 | Servlet Path /Tes
tFilterServlet | after request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter2 | Servlet Path /Tes
tFilterServlet | after request
10-Jan-2018 12:16:49.252 INFO [http-nio-10599-exec-2] org.apache.catalina.core.ApplicationContext.log TestFilter1 | Servlet Path /Tes
tFilterServlet | after request
```

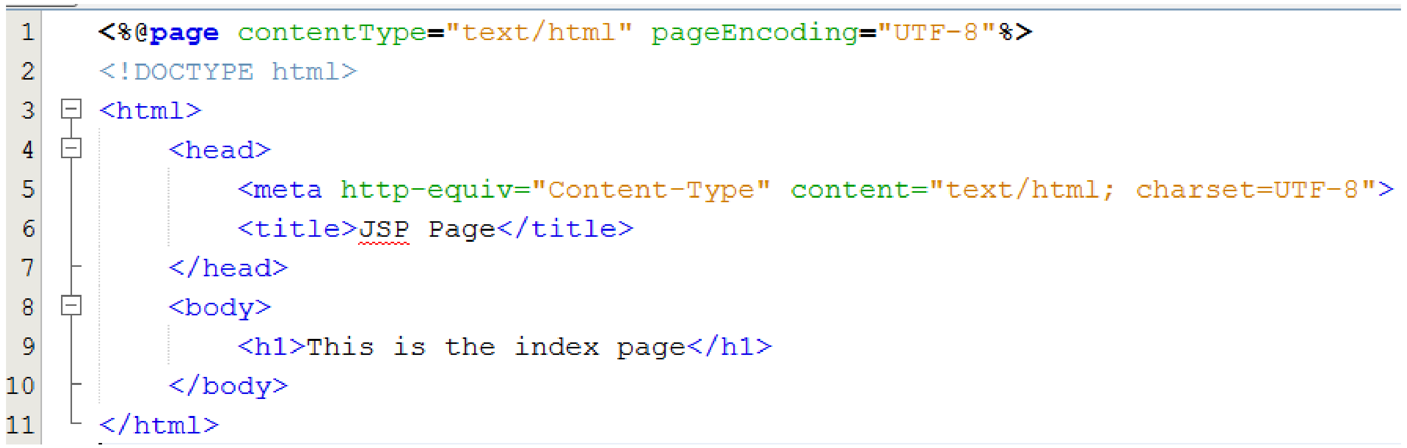
Code for TestFilterServlet



```java
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet TestFilterServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet TestFilterServlet at " + request.getContextPath() + "</h1>");
            out.println("</body>");
            out.println("</html>");

            String url = "/secondpage.jsp";
            RequestDispatcher dispatcher = request.getRequestDispatcher(url);
            dispatcher.forward(request, response);
        } finally {
            out.close();
        }
    }
```

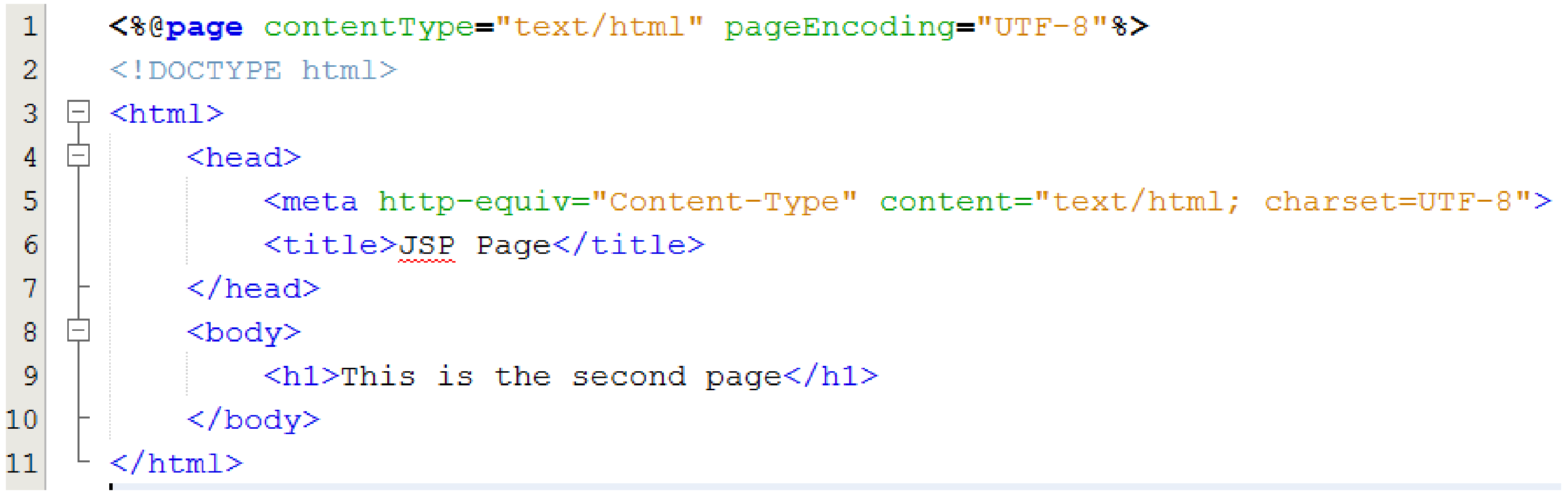
Code for the JSP Files in the Example

index.jsp



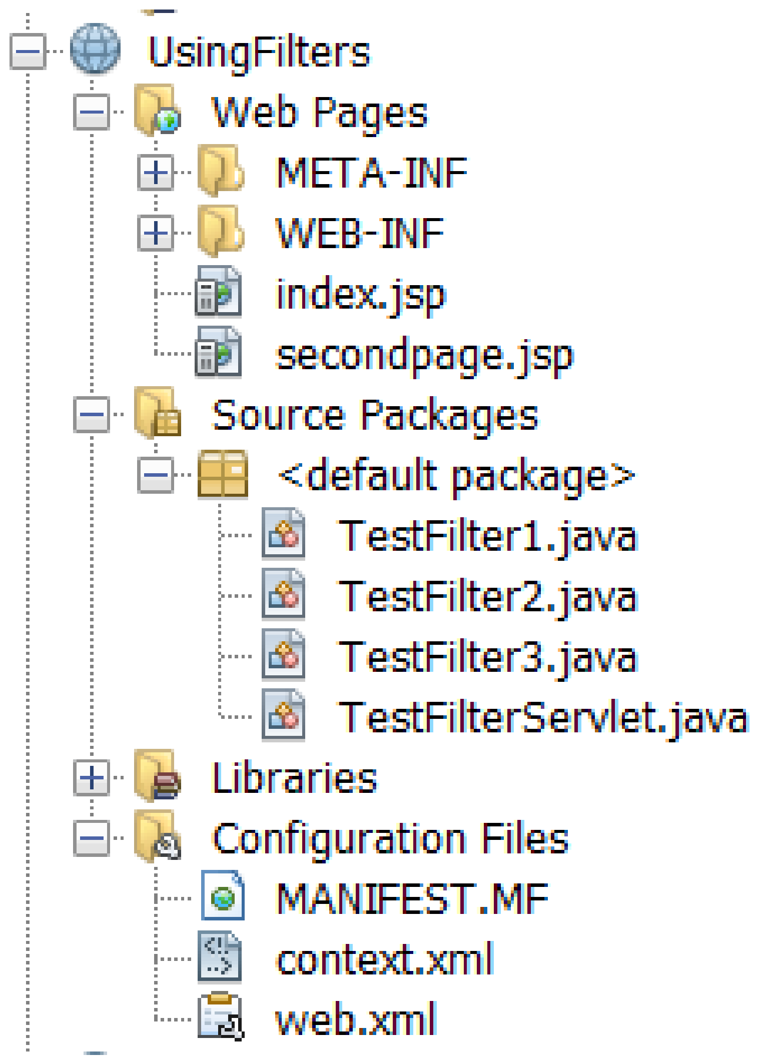
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>This is the index page</h1>
    </body>
</html>
```

secondpage.jsp



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>This is the second page</h1>
    </body>
</html>
```

Anatomy of the project



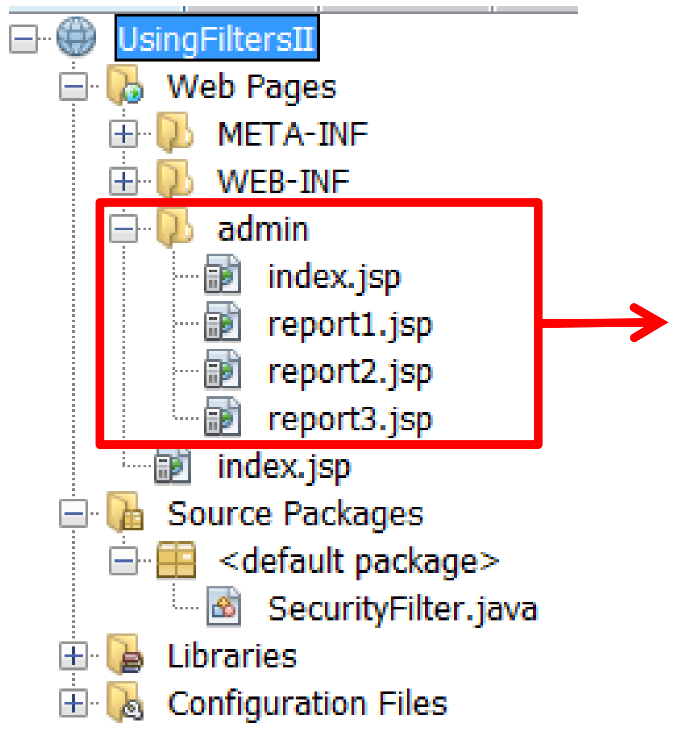
```
UsingFilters
    Web Pages
        META-INF
        WEB-INF
        index.jsp
        secondpage.jsp
    Source Packages
        <default package>
            TestFilter1.java
            TestFilter2.java
            TestFilter3.java
            TestFilterServlet.java
    Libraries
    Configuration Files
        MANIFEST.MF
        context.xml
        web.xml
```

Real World Example

- Usernames and passwords are a good first line of defence for protecting access to your web site.
  - Sometimes though you might want an additional level of security.
  - For example, you might want to restrict access to the admin section of your application to only a few IP addresses such as local addresses.
- The upcoming example shows how this can be achieved relatively easily.

Real World Example

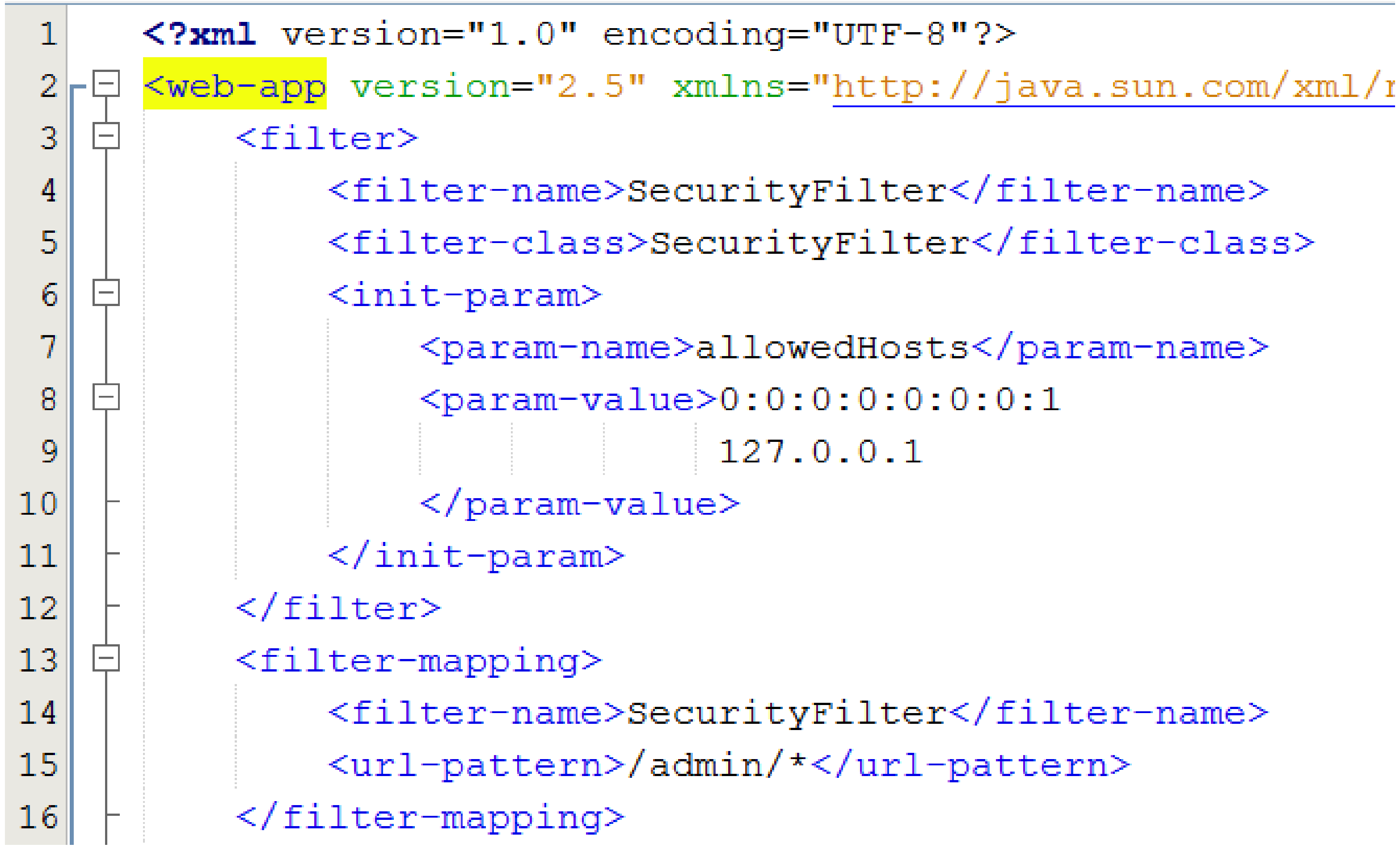
- Anatomy of the project:



```
UsingFiltersII
    Web Pages
        META-INF
        WEB-INF
        admin
            index.jsp
            report1.jsp
            report2.jsp
            report3.jsp
        index.jsp
    Source Packages
        <default package>
            SecurityFilter.java
    Libraries
    Configuration Files
```

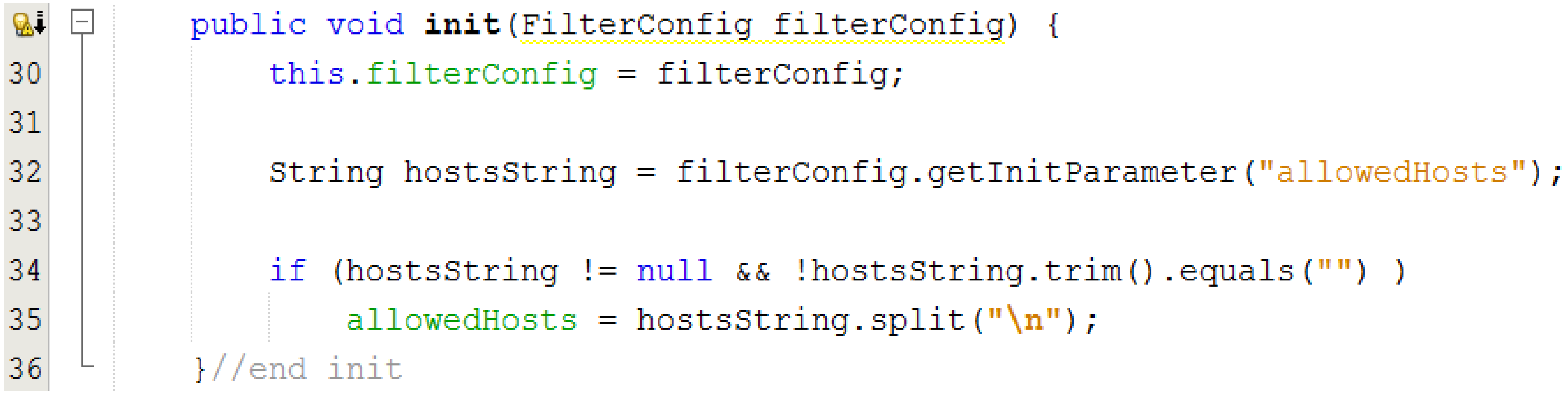
All pages within the admin folder are only available to those who contact the application with a local IP address.

web.xml



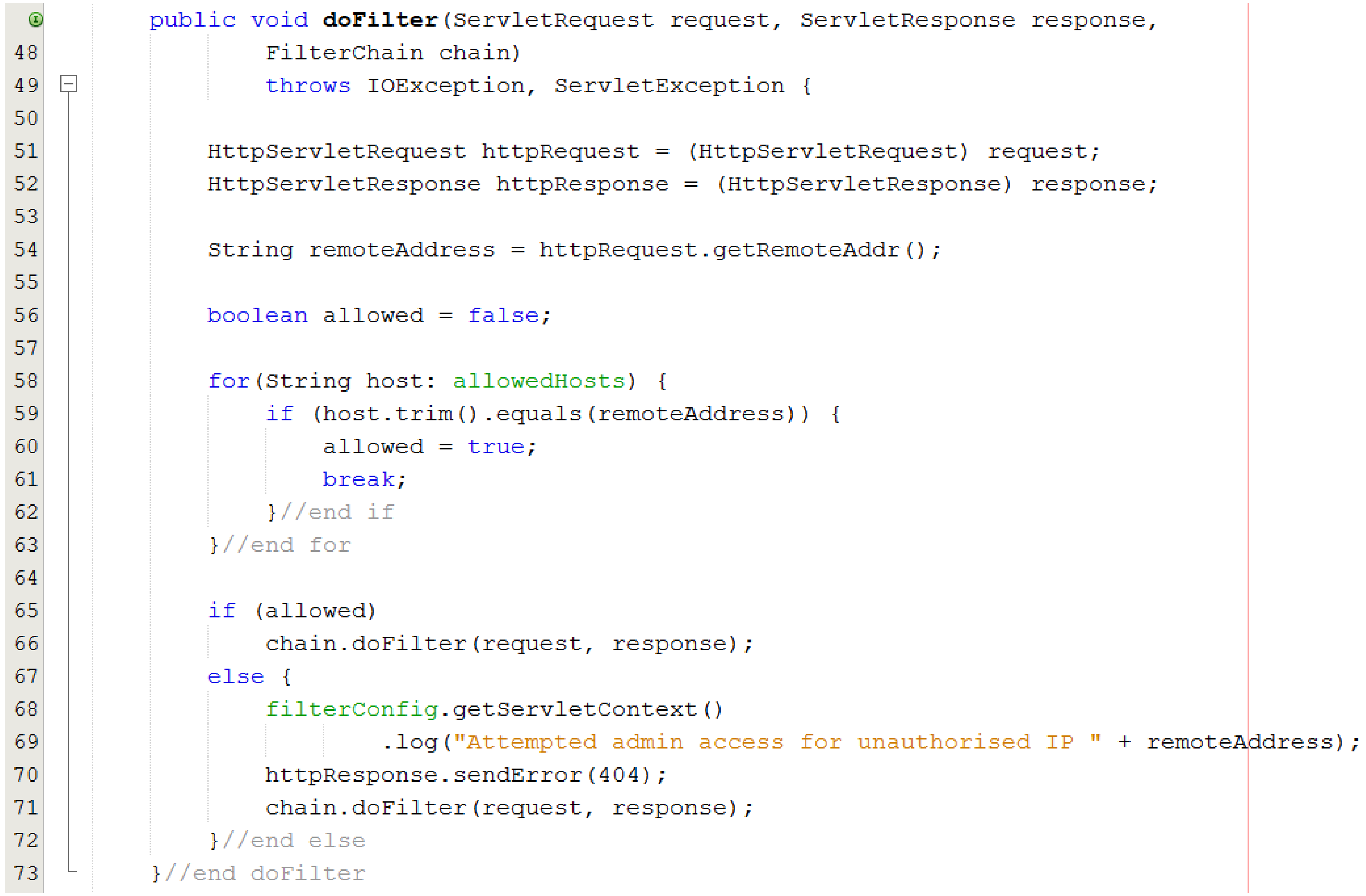
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/r
    <filter>
        <filter-name>SecurityFilter</filter-name>
        <filter-class>SecurityFilter</filter-class>
        <init-param>
            <param-name>allowedHosts</param-name>
            <param-value>0:0:0:0:0:0:0:1
                        127.0.0.1
            </param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>SecurityFilter</filter-name>
        <url-pattern>/admin/*</url-pattern>
    </filter-mapping>
```

SecurityFilter – init method



```
    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;

        String hostsString = filterConfig.getInitParameter("allowedHosts");

        if (hostsString != null && !hostsString.trim().equals("") )
            allowedHosts = hostsString.split("\n");
    }//end init
```

SecurityFilter – doFilter method



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

    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpServletResponse httpResponse = (HttpServletResponse) response;

    String remoteAddress = httpRequest.getRemoteAddr();

    boolean allowed = false;

    for(String host: allowedHosts) {
        if (host.trim().equals(remoteAddress)) {
            allowed = true;
            break;
        }//end if
    }//end for

    if (allowed)
        chain.doFilter(request, response);
    else {
        filterConfig.getServletContext()
                .log("Attempted admin access for unauthorised IP " + remoteAddress);
        httpResponse.sendError(404);
        chain.doFilter(request, response);
    }//end else
}//end doFilter
```

References

Murach, J., (2014) *Murachs Java Servlets JSP*, 3rd edn. Mike Murach and Associates, Inc.