```
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy()
    {
        // do nothing.
    }
}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows:

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable.

If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory. Next section would explain how a compiled servlet would be deployed in production.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installation-directory>/webapps/ROOT and the class file would reside in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
       String title = "Reading All Form Parameters";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
        "<tr bgcolor=\"#949494\">\n" +
        "<th>Param Name</th><th>Param Value(s)</th>\n"+
        "</tr>\n");

      Enumeration paramNames = request.getParameterNames();

      while(paramNames.hasMoreElements()) {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues =
```

Methods to Set HTTP Response Header

There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with *HttpServletResponse* object.

S.N.	Method & Description
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)** Adds a response header with the given name and date-value.
7	**void addHeader(String name, String value)** Adds a response header with the given name and value.
8	**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9	**void flushBuffer()** Forces any content in the buffer to be written to the client.
10	**void reset()** Clears any data that exists in the buffer as well as the status code and headers.
11	**void resetBuffer()** Clears the content of the underlying buffer in the response without clearing headers or status code.
12	**void sendError(int sc)** Sends an error response to the client using the specified status code and clearing the buffer.
13	**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14	**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15	**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.
16	**void setCharacterEncoding(String charset)**

8. Servlets – Http Status Codes

The format of the HTTP request and HTTP response messages are similar and will have following structure:

- An initial status line + CRLF (Carriage Return + Line Feed i.e. New Line)
- Zero or more header lines + CRLF
- A blank line, i.e., a CRLF
- An optional message body like file, query data or query output.

For example, a server response header looks as follows:

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
  (Blank Line)
<!doctype ...>
<html>
<head>...</head>
<body>
...
</body>
</html>
```


The status line consists of the HTTP version (HTTP/1.1 in the example), a status code (200 in the example), and a very short message corresponding to the status code (OK in the example).

Following is a list of HTTP status codes and associated messages that might be returned from the Web Server:

Code	Message	Description
100	Continue	Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request
101	Switching Protocols	The server switches protocol.

200	OK	The request is OK
201	Created	The request is complete, and a new resource is created
202	Accepted	The request is accepted for processing, but the processing is not complete.
203	Non-authoritative Information	
204	No Content	
205	Reset Content	
206	Partial Content	
300	Multiple Choices	A link list. The user can select a link and go to that location. Maximum five addresses
301	Moved Permanently	The requested page has moved to a new url
302	Found	The requested page has moved temporarily to a new url
303	See Other	The requested page can be found under a different url
304	Not Modified	
305	Use Proxy	
306	*Unused*	This code was used in a previous version. It is no longer used, but the code is reserved.
307	Temporary Redirect	The requested page has moved temporarily to a new url.
400	Bad Request	The server did not understand the request
401	Unauthorized	The requested page needs a username and a password
402	Payment Required	*You cannot use this code yet*
403	Forbidden	Access is forbidden to the requested page
404	Not Found	The server cannot find the requested page.
405	Method Not Allowed	The method specified in the request is not allowed.

3	**javax.servlet.error.message** This attribute gives information exact error message which can be stored and analyzed after storing in a java.lang.String data type.
4	**javax.servlet.error.request_uri** This attribute gives information about URL calling the servlet and it can be stored and analysed after storing in a java.lang.String data type.
5	**javax.servlet.error.exception** This attribute gives information about the exception raised, which can be stored and analysed .
6	**javax.servlet.error.servlet_name** This attribute gives servlet name which can be stored and analyzed after storing in a java.lang.String data type.

Error Handler Servlet – Example

This example would give you basic understanding of Exception Handling in Servlet, but you can write more sophisticated filter applications using the same concept:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ErrorHandler extends HttpServlet {

  // Method to handle GET method request.
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Analyze the servlet exception
      Throwable throwable = (Throwable)
      request.getAttribute("javax.servlet.error.exception");
      Integer statusCode = (Integer)
      request.getAttribute("javax.servlet.error.status_code");
      String servletName = (String)
      request.getAttribute("javax.servlet.error.servlet_name");
      if (servletName == null){
         servletName = "Unknown";
```

```
         out.println("The exception message: " +
                                   throwable.getMessage( ));
      }
      out.println("</body>");
      out.println("</html>");
  }
  // Method to handle POST method request.
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
     doGet(request, response);
  }
}
```

Compile **ErrorHandler.java** in usual way and put your class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

Let us add the following configuration in web.xml to handle exceptions:

```
<servlet>
        <servlet-name>ErrorHandler</servlet-name>
        <servlet-class>ErrorHandler</servlet-class>
</servlet>
<!-- servlet mappings -->
<servlet-mapping>
        <servlet-name>ErrorHandler</servlet-name>
        <url-pattern>/ErrorHandler</url-pattern>
</servlet-mapping>
<error-page>
    <error-code>404</error-code>
    <location>/ErrorHandler</location>
</error-page>
<error-page>
    <exception-type>java.lang.Throwable</exception-type >
    <location>/ErrorHandler</location>
</error-page>
```


Now try to use a servlet which raise any exception or type a wrong URL, this would trigger Web Container to call **ErrorHandler** servlet and display an appropriate message

	This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

Setting Cookies with Servlet

Setting cookies with servlet involves three steps:

(1) Creating a Cookie object: You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Keep in mind, neither the name nor the value should contain white space or any of the following characters:

```
[ ] ( ) = , " / ? @ : ;
```

(2) Setting the maximum age: You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60*60*24);
```

(3) Sending the Cookie into the HTTP response headers: You use response.addCookie to add cookies in the HTTP response header as follows:

```
response.addCookie(cookie);
```


Example

Let us modify our FormExample to set the cookies for first and last name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Create cookies for first and last names.

      Cookie firstName = new Cookie("first_name",
```

```
                        request.getParameter("first_name"));
      Cookie lastName = new Cookie("last_name",
                        request.getParameter("last_name"));

      // Set expiry date after 24 Hrs for both the cookies.
      firstName.setMaxAge(60*60*24);
      lastName.setMaxAge(60*60*24);

      // Add both the cookies in the response header.
      response.addCookie( firstName );
      response.addCookie( lastName );

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Setting Cookies Example";
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
                  "<html>\n" +
                  "<head><title>" + title + "</title></head>\n" +
                  "<body bgcolor=\"#f0f0f0\">\n" +
                  "<h1 align=\"center\">" + title + "</h1>\n" +
                  "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
                  "</ul>\n" +
                  "</body></html>");
   }
}
```


Compile the above servlet **HelloForm** and create appropriate entry in web.xml file and finally try following HTML page to call servlet.

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

```
File Upload:

Select a file to upload:

Choose File        No File Chosen

Upload File

NOTE: This is just dummy form and would not work.
```

Writing Backend Servlet

Following is the servlet **UploadServlet** which would take care of accepting uploaded file and to store it in directory <Tomcat-installation-directory>/webapps/data. This directory name could also be added using an external configuration such as a **context-param** element in web.xml as follows:

```
<web-app>

....

<context-param>

    <description>Location to store uploaded file</description>

    <param-name>file-upload</param-name>

    <param-value>

         c:\apache-tomcat-8.0.28\webapps\data\

     </param-value>

</context-param>

....

</web-app>
```


Following is the source code for UploadServlet which can handle multiple file uploading at a time. Before proceeding you have make sure the followings:

- Following example depends on FileUpload, so make sure you have the latest version of **commons-fileupload.x.x.jar** file in your classpath. You can download it from http://commons.apache.org/fileupload/.

- FileUpload depends on Commons IO, so make sure you have the latest version of **commons-io-x.x.jar** file in your classpath. You can download it from http://commons.apache.org/io/.

```
      out.println("</head>");
      out.println("<body>");
      while ( i.hasNext () )
      {
         FileItem fi = (FileItem)i.next();
         if ( !fi.isFormField () )
         {
            // Get the uploaded file parameters
            String fieldName = fi.getFieldName();
            String fileName = fi.getName();
            String contentType = fi.getContentType();
            boolean isInMemory = fi.isInMemory();
            long sizeInBytes = fi.getSize();
            // Write the file
            if( fileName.lastIndexOf("\\") >= 0 ){
               file = new File( filePath +
               fileName.substring( fileName.lastIndexOf("\\"))) ;
            }else{
               file = new File( filePath +
               fileName.substring(fileName.lastIndexOf("\\")+1)) ;
            }
            fi.write( file ) ;
            out.println("Uploaded Filename: " + fileName + "<br>");
         }
      }
      out.println("</body>");
      out.println("</html>");
   }catch(Exception ex) {
       System.out.println(ex);
   }
   }
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)

        throws ServletException, java.io.IOException {


        throw new ServletException("GET method used with " +
```

```
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Display Current Date & Time";
      Date dNow = new Date( );
      SimpleDateFormat ft =
      new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
      String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
      out.println(docType +
        "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n" +
        "<h1 align=\"center\">" + title + "</h1>\n" +
        "<h2 align=\"center\">" + ft.format(dNow) + "</h2>\n" +
        "</body></html>");
  }
}
```


Compile above servlet once again and then call this servlet using URL http://localhost:8080/CurrentDate. This would produce following result:

Display Current Date & Time

Mon 2010.06.21 at 10:06:44 PM GMT+04:00

Simple DateFormat Format Codes

To specify the time format use a time pattern string. In this pattern, all ASCII letters are reserved as pattern letters, which are defined as the following:

```
 <servlet>
     <servlet-name>PageHitCounter</servlet-name>
     <servlet-class>PageHitCounter</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>PageHitCounter</servlet-name>
     <url-pattern>/PageHitCounter</url-pattern>
 </servlet-mapping>
....
```

Now call this servlet using URL http://localhost:8080/PageHitCounter. This would increase counter by one every time this page gets refreshed and it would display following result:

```
Total Number of Hits

6

Hit Counter for a Website
```


Many times you would be interested in knowing total number of hits on your whole website. This is also very simple in Servlet and we can achieve this using filters.

Following are the steps to be taken to implement a simple website hit counter which is based on Filter Life Cycle:

- Initialize a global variable in init() method of a filter.
- Increase global variable every time doFilter method is called.
- If required, you can use a database table to store the value of global variable in destroy() method of filter. This value can be read inside init() method when filter would be initialized next time. This step is optional.

Here I'm assuming that the web container will not be restarted. If it is restarted or servlet destroyed, the hit counter will be reset.

Example

This example shows how to implement a simple website hit counter:

```
// Import required java libraries
import java.io.*;
```

```
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";
         out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor=\"#f0f0f0\">\n" +
         "<h1 align=\"center\">" + title + "</h1>\n" +
         "<p align=\"center\">" + res + "</p>\n" +
         "</body></html>");
      }catch (MessagingException mex) {
         mex.printStackTrace();
      }
   }
}
```

Now let us compile the above servlet and create the following entries in web.xml

```
....
 <servlet>
     <servlet-name>SendEmail</servlet-name>
     <servlet-class>SendEmail</servlet-class>
 </servlet>

 <servlet-mapping>
     <servlet-name>SendEmail</servlet-name>
     <url-pattern>/SendEmail</url-pattern>
 </servlet-mapping>
....
```


Now call this servlet using URL http://localhost:8080/SendEmail which would send an email to given email ID *abcd@gmail.com* and would display following response:

```
                    Send Email

Sent message successfully....
```

If you want to send an email to multiple recipients then following methods would be used to specify multiple email IDs:

```
void addRecipients(Message.RecipientType type,
```

```
String from = "web@gmail.com";

// Assuming you are sending email from localhost
String host = "localhost";

// Get system properties
Properties properties = System.getProperties();

// Setup mail server
properties.setProperty("mail.smtp.host", host);

// Get the default Session object.
Session session = Session.getDefaultInstance(properties);

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

try{
   // Create a default MimeMessage object.
   MimeMessage message = new MimeMessage(session);
   // Set From: header field of the header.
   message.setFrom(new InternetAddress(from));
   // Set To: header field of the header.
   message.addRecipient(Message.RecipientType.TO,
                            new InternetAddress(to));
   // Set Subject: header field
   message.setSubject("This is the Subject Line!");

   // Send the actual HTML message, as big as you like
   message.setContent("<h1>This is actual message</h1>",
                      "text/html" );
   // Send message
   Transport.send(message);

   String title = "Send Email";

   String res = "Sent message successfully....";
```

20. Servlets – Packaging

The web application structure involving the WEB-INF subdirectory is standard to all Java web applications and specified by the servlet API specification. Given a top-level directory name of myapp. Here is how this directory structure looks like:

```
/myapp
    /images
    /WEB-INF
        /classes
        /lib
```

The WEB-INF subdirectory contains the application's deployment descriptor, named web.xml. All the HTML files should be kept in the top-level directory which is *myapp*. For admin user, you would find ROOT directory as parent directory.

Creating Servlets in Packages

The WEB-INF/classes directory contains all the servlet classes and other class files, in a structure that matches their package name. For example, if you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in the following directory:

```
/myapp/WEB-INF/classes/com/myorg/MyServlet.class
```

Following is the example to create MyServlet class with a package name *com.myorg*

```
// Name your package
package com.myorg;

// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

  private String message;

  public void init() throws ServletException
```

very similar to how applets are debugged. The difference is that with applets, the actual program being debugged is sun.applet.AppletViewer.

Most debuggers hide this detail by automatically knowing how to debug applets. Until they do the same for servlets, you have to help your debugger by doing the following:

- Set your debugger's classpath so that it can find sun.servlet.http.Http-Server and associated classes.

- Set your debugger's classpath so that it can also find your servlets and support classes, typically server_root/servlets and server_root/classes.

You normally wouldn't want server_root/servlets in your classpath because it disables servlet reloading. This inclusion, however, is useful for debugging. It allows your debugger to set breakpoints in a servlet before the custom servlet loader in HttpServer loads the servlet.

Once you have set the proper classpath, start debugging sun.servlet.http.HttpServer. You can set breakpoints in whatever servlet you're interested in debugging, then use a web browser to make a request to the HttpServer for the given servlet (http://localhost:8080/servlet/ServletToDebug). You should see execution being stopped at your breakpoints.

Using Comments

Comments in your code can help the debugging process in various ways. Comments can be used in lots of other ways in the debugging process.

The Servlet uses Java comments and single line (// ...) and multiple line (/* ... */) comments can be used to temporarily remove parts of your Java code. If the bug disappears, take a closer look at the code you just commented and find out the problem.

Client and Server Headers

Sometimes when a servlet doesn't behave as expected, it's useful to look at the raw HTTP request and response. If you're familiar with the structure of HTTP, you can read the request and response and see exactly what exactly is going with those headers.

Important Debugging Tips

Here is a list of some more debugging tips on servlet debugging:

- Remember that server_root/classes doesn't reload and that server_root/servlets probably does.

- Ask a browser to show the raw content of the page it is displaying. This can help identify formatting problems. It's usually an option under the View menu.

- Make sure the browser isn't caching a previous request's output by forcing a full reload of the page. With Netscape Navigator, use Shift-Reload; with Internet Explorer use Shift-Refresh.

Languages Setting

A servlet can output a page written in a Western European language such as English, Spanish, German, French, Italian, Dutch etc. Here it is important to set Content-Language header to display all the characters properly.

Second point is to display all the special characters using HTML entities. For example, "ñ" represents "ñ", and "¡" represents "¡" as follows:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Locale;

public class DisplaySpanish extends HttpServlet{

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
            throws ServletException, IOException
  {
    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    // Set spanish language code.
    response.setHeader("Content-Language", "es");

    String title = "En Espa&ntilde;ol";
    String docType =
     "<!doctype html public \"-//w3c//dtd html 4.0 " +
     "transitional//en\">\n";
     out.println(docType +
     "<html>\n" +
     "<head><title>" + title + "</title></head>\n" +
     "<body bgcolor=\"#f0f0f0\">\n" +
     "<h1>" + "En Espa&ntilde;ol:" + "</h1>\n" +
     "<h1>" + "&iexcl;Hola Mundo!" + "</h1>\n" +
     "</body></html>");
  }
```

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(value = "/Simple", initParams = {
        @WebInitParam(name="foo", value="Hello "),
        @WebInitParam(name="bar", value=" World!")
     })
public class Simple extends HttpServlet {
     private static final long serialVersionUID = 1L;

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

        out.print("<html><body>");
        out.print("<h3>Hello Servlet</h3>");
        out.println(getInitParameter("foo"));
        out.println(getInitParameter("bar"));
        out.print("</body></html>");
    }
}
```


Compile **Simple.java** in the usual way and put your class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

Now try to call any servlet by just running *http://localhost:8080/Simple*. You will see the following output on the web page.

```
Hello Servlet


Hello World!
```

@Webfilter

This is the annotation used to declare a servlet filter. It is processed by the container at deployment time, and the corresponding filter applied to the specified URL patterns, servlets, and dispatcher types.

The **@WebFilter** annotation defines a filter in a web application. This annotation is specified on a class and contains metadata about the filter being declared. The annotated filter must specify at least one URL pattern. The following table lists the attributes used for WebFilter annotation.

Attribute Name	Description
String filterName	Name of the filter
String[] value Or String[] urlPatterns	Provides array of values or urlPatterns to which the filter applies
DispatcherType[] dispatcherTypes	Specifies the types of dispatcher (Request/Response) to which the filter applies
String[] servletNames	Provides an array of servlet names
String displayName	Name of the filter
String description	Description of the filter
WebInitParam[] initParams	Array of initialization parameters for this filter
Boolean asyncSupported	Asynchronous operation supported by this filter
String smallIcon	Small icon for this filter, if present
String largeIcon	Large icon for this filter, if present

Example

The following example describes how to use @WebFilter annotation. It is a simple LogFilter that displays the value of Init-param **test-param** and the current time timestamp on the console. That means, the filter works like an interface layer between the request and the response. Here we use "/*" for urlPattern. It means, this filter is applicable for all the servlets.

```
import java.io.IOException;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.*;
import java.util.*;

// Implements Filter class
```