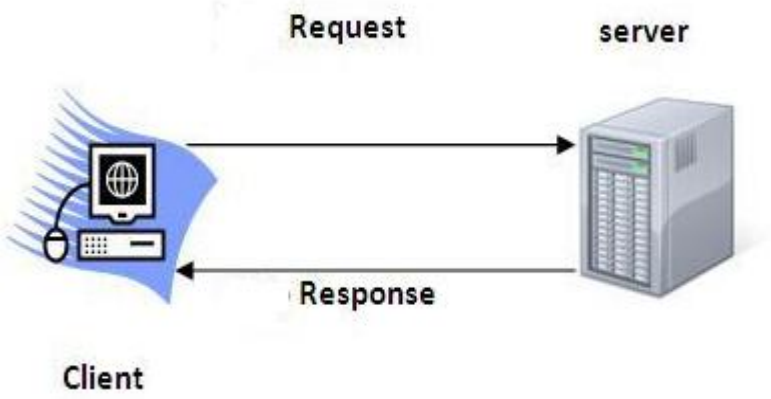
Servlet Programming

By
Apparao G

Introduction:

A servlet is a Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, using HTTP Protocol.



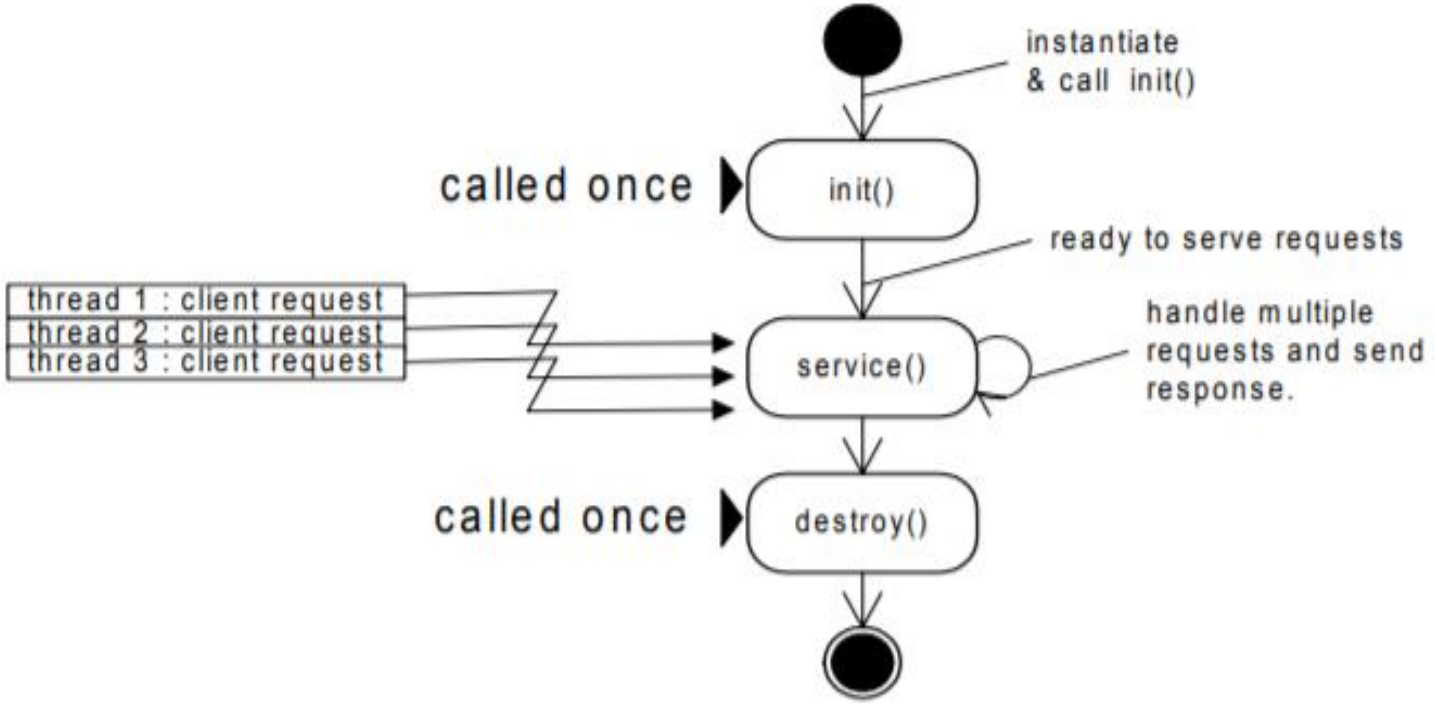
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

To implement servlet interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.

Servlet Life Cycle:

Servlet life cycle consists of 3 major life cycle methods.

1. initialization of servlet
2. Serving the client request
3. destroying the servlet instance.



The Web container is responsible for managing the servlet’s life cycle.

init() method:

The Web container creates an instance of the servlet and then the container calls the init() method.

At the completion of the init() method the servlet is in ready state to service requests from clients.

This method is called only once.

Signature:

public void init(ServletConfig config) throws ServletException

Parameters:

config - a ServletConfig object containing the servlet's configuration and initialization parameters

Throws:

ServletException - if an exception has occurred that interferes with the servlet's normal operation

service() method:

The container calls the servlet’s service() method for handling each request by spawning a new thread for each request from the Web container’s thread pool (It is also possible to have a single threaded Servlet).

Signature:

public void service(ServletRequest req,ServletResponse res)

throws ServletException,java.io.IOException

Parameters:

req - the ServletRequest object that contains the client's request

res - the ServletResponse object that contains the servlet's response

Throws:

ServletException - if an exception occurs that interferes with the servlet's normal operation

java.io.IOException - if an input or output exception occurs

destroy() method:

Before destroying the instance the container will call the destroy() method. After destroy() the servlet becomes the potential candidate for garbage collection.

This method is only called once all threads within the servlet's service method have exited or after a timeout period has passed. After the servlet container calls this method, it will not call the service method again on this servlet.

This method gives the servlet an opportunity to clean up any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the servlet's current state in memory.

Signature:

public void destroy()

getServletConfig:

Signature:

public ServletConfig getServletConfig()

Returns a ServletConfig object, which contains initialization and startup parameters for this servlet. The ServletConfig object returned is the one passed to the init method.

Implementations of this interface are responsible for storing the ServletConfig object so that this method can return it. The GenericServlet class, which implements this interface, already does this.

Returns:

the ServletConfig object that initializes this servlet

getServletInfo:

Signature:

public java.lang.String getServletInfo()

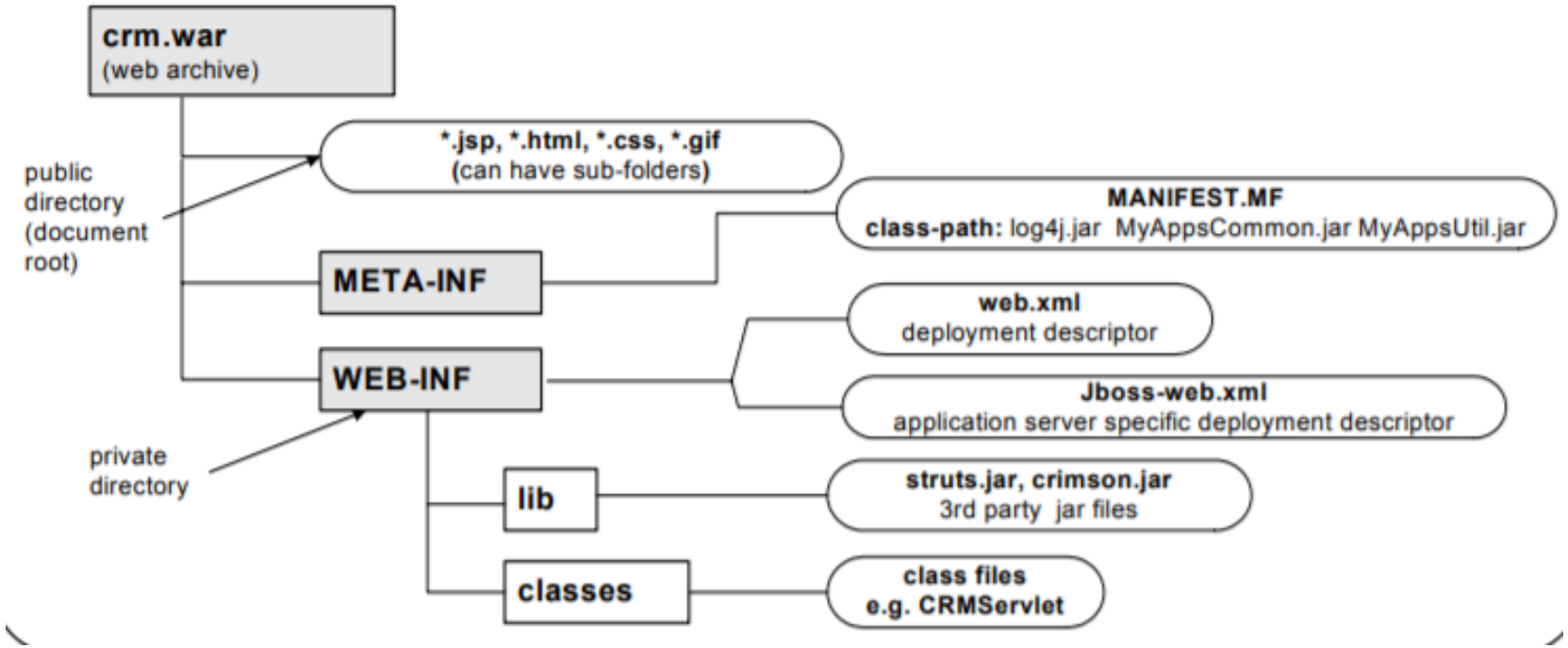
Returns information about the servlet, such as author, version, and copyright.

The string that this method returns should be plain text and not markup of any kind (such as HTML, XML, etc.).

Returns:

a String containing servlet information

Directory structure of a web application:



A public resource directory (document root): The document root is where JSP pages, client-side classes

and archives, and static Web resources are stored.

A private directory called WEB-INF: which contains following files and directories:

- web.xml: Web application deployment descriptor.
- application server specific deployment descriptor e.g. jboss-web.xml etc.
- *.tld: Tag library descriptor files.
- classes: A directory that contains server side classes like servlets, utility classes, JavaBeans etc.
- lib: A directory where JAR (archive files of tag libraries, utility libraries used by the server side classes) files are stored.

To implement servlet interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.

GenericServlet:

- Defines a generic, protocol-independent servlet.
- GenericServlet implements the Servlet and ServletConfig interfaces.
- GenericServlet makes writing servlets easier. It provides simple versions of the lifecycle methods init and destroy and of the methods in the ServletConfig interface. GenericServlet also implements the log method, declared in the ServletContext interface.
- To write a generic servlet, you need only override the abstract service method.

```
public void service(ServletRequest req,ServletResponse res)
        throws ServletException,java.io.IOException
```

GenericServlet Example:

Follow below steps for creating web project:

1. open eclipse
2. go to File --> New --> select "Dynamic Web Project"
3. Fill below values
   Project name: Calculator
   Dynamic web module version: 3.0 0r later version
   click Next
   select Generate web.xml deployment descriptor

4. Create index.html file under "WebContent" folder

```
<!DOCTYPE html>
<html>
<body>
		<form action="add">
			Enter first number  : <input type="text" name="num1" id="num1"></br>
			Enter second number : <input type="text" name="num2" id="num2"></br>
			<input type="submit" value="Add">
		</form>
</body>
</html>
```

5. Create AddServlet.java class under src folder

```
public class AddServlet extends GenericServlet {

        private static final long serialVersionUID = 1L;

        @Override
        public void service(ServletRequest request, ServletResponse response) throws ServletException,
                IOException {

                int num1 = Integer.parseInt(request.getParameter("num1"));
                int num2 = Integer.parseInt(request.getParameter("num2"));
                int result = num1 + num2;
                System.out.println("Addition of two numbers:"+result);
                PrintWriter pw = response.getWriter();
                pw.println("Result ::"+result);
        }

}
```

6. Register the servlet in web.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
id="WebApp_ID" version="4.0">
  <display-name>Calculator</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
            <servlet-name>addServlet</servlet-name>
           <servlet-class>com.test.servlet.AddServlet</servlet-class>
  </servlet>
  <servlet-mapping>
            <servlet-name>addServlet</servlet-name>
            <url-pattern>/add</url-pattern>
  </servlet-mapping>
</web-app>
```

Steps to deploy and run the web application:

1. Go to servers tab

   click on "Click this link to create a new server..." it open the popup

   select the Apache --> Tomcat v9.0 Server

   click Next button

   Selec the "Calculator" project from left side grid and click on Add> option

   click on "Finish" button

2. Go to servers tab

   Right click on "Tomcat v9.0 Server at localhost"

   Select "Start" option

3. Open the you favorite browser paste the below link and hit enter button

   http://localhost:8080/Calculator/

ServletRequest Interface:

Defines an object to provide client request information to a servlet. The servlet container creates a ServletRequest object and passes it as an argument to the servlet's service method.

A ServletRequest object provides data including parameter name and values, attributes, and an input stream. Interfaces that extend ServletRequest can provide additional protocol-specific data (for example, HTTP data is provided by HttpServletRequest.

Frequently used Method:

getParameter method:

public java.lang.String getParameter(java.lang.String name)

Returns the value of a request parameter as a String, or null if the parameter does not exist. Request parameters are extra information sent with the request. For HTTP servlets, parameters are contained in the query string or posted form data.

You should only use this method when you are sure the parameter has only one value. If the parameter might have more than one value, use getParameterValues(java.lang.String).

getParameterValues method:

public java.lang.String[] getParameterValues(java.lang.String name)

Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.

If the parameter has a single value, the array has a length of 1.

getParameterMap method:

public java.util.Map getParameterMap()

Returns a java.util.Map of the parameters of this request. Request parameters are extra information sent with the request. For HTTP servlets, parameters are contained in the query string or posted form data.

getServerName Method:

public java.lang.String getServerName()

Returns the host name of the server to which the request was sent. It is the value of the part before ":" in the Host header value, if any, or the resolved server name, or the server IP address.

getServerPort Method:

public int getServerPort()

Returns the port number to which the request was sent. It is the value of the part after ":" in the Host header value, if any, or the server port where the client connection was accepted on.

getRemoteAddr Method:

public java.lang.String getRemoteAddr()

Returns the Internet Protocol (IP) address of the client or last proxy that sent the request.

ServletResponse Interface:

Defines an object to assist a servlet in sending a response to the client. The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.

To send binary data in a MIME body response, use the ServletOutputStream returned by getOutputStream(). To send character data, use the PrintWriter object returned by getWriter(). To mix binary and text data, for example, to create a multipart response, use a ServletOutputStream and manage the character sections manually.

Frequently used methods:

setContentType method:

public void setContentType(java.lang.String type)

Sets the content type of the response being sent to the client, if the response has not been committed yet. The given content type may include a character encoding specification, for example, text/html;charset=UTF-8. The response's character encoding is only set from the given content type if this method is called before getWriter is called.

This method may be called repeatedly to change content type and character encoding. This method has no effect if called after the response has been committed. It does not set the response's character encoding if it is called after getWriter has been called or after the response has been committed.

getContentType method:

public java.lang.String getContentType()

Returns the content type used for the MIME body sent in this response.

For example, text/html; charset=UTF-8, or null

getOutputStream method:

public ServletOutputStream getOutputStream() throws java.io.IOException

Returns a ServletOutputStream suitable for writing binary data in the response. The servlet container does not encode the binary data.

Calling flush() on the ServletOutputStream commits the response.

getWriter method:

public java.io.PrintWriter getWriter() throws java.io.IOException

Returns a PrintWriter object that can send character text to the client. The PrintWriter uses the character encoding returned by getCharacterEncoding().

Calling flush() on the PrintWriter commits the response.

setLocale method:

public void setLocale(java.util.Locale loc)

Sets the locale of the response, if the response has not been committed yet.

getLocale method:

public java.util.Locale getLocale()

Returns the locale specified for response.

HttpServlet: It provides functionality related to HTTP protocol.

It extends GenericServlet

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of HttpServlet must override at least one method, usually one of these:

- doGet, if the servlet supports HTTP GET requests
- doPost, for HTTP POST requests
- doPut, for HTTP PUT requests
- doDelete, for HTTP DELETE requests
- init and destroy, to manage resources that are held for the life of the servlet
- getServletInfo, which the servlet uses to provide information about itself

Note:Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections.

Method Summary

protected void	**doDelete**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a DELETE request.
protected void	**doGet**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a GET request.
protected void	**doHead**(HttpServletRequest req, HttpServletResponse resp) Receives an HTTP HEAD request from the protected service method and handles the request.
protected void	**doOptions**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a OPTIONS request.
protected void	**doPost**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a POST request.
protected void	**doPut**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a PUT request.
protected void	**doTrace**(HttpServletRequest req, HttpServletResponse resp) Called by the server (via the service method) to allow a servlet to handle a TRACE request.
protected long	**getLastModified**(HttpServletRequest req) Returns the time the HttpServletRequest object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	**service**(HttpServletRequest req, HttpServletResponse resp) Receives standard HTTP requests from the public service method and dispatches them to the do*XXX* methods defined in this class.
void	**service**(ServletRequest req, ServletResponse res) Dispatches client requests to the protected service method.

HttpServlet Example:

Create web project:

1. open eclipse
2. go to File --> New --> select "Dynamic Web Project"
3. Fill below values
   Project name: MathsApp
   Dynamic web module version: 3.0 0r later version
   click Next
   select Generate web.xml deployment descriptor

4. Create index.html file under "WebContent" folder

```
<!DOCTYPE html>
<html>
<body>
		<form action="sub">
			Enter first number  : <input type="text" name="num1" id="num1"></br>
			Enter second number : <input type="text" name="num2" id="num2"></br>
			<input type="submit" value="Substraction">
		</form>
</body>
</html>
```

5. Create SubtractServlet.java class under src folder

```
public class SubstractionServlet extends HttpServlet{

		private static final long serialVersionUID = 1L;

		@Override
		public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
				int num1 = Integer.parseInt(request.getParameter("num1"));
				int num2 = Integer.parseInt(request.getParameter("num2"));
				int result = num1 - num2;
				System.out.println("Substraction of two numbers:"+result);
				 response.setContentType("text/html");
				PrintWriter pw = response.getWriter();
				pw.println("Substraction of two numbers ::"+result);
		}

}
```

6. Register the servlet in web.xml file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID"
version="4.0">
  <display-name>Calculator</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
                <servlet-name>subServlet</servlet-name>
                <servlet-class>com.test.servlet.SubstractionServlet</servlet-class>
  </servlet>
  <servlet-mapping>
                                <servlet-name>subServlet</servlet-name>
                                <url-pattern>/sub</url-pattern>
  </servlet-mapping>
</web-app>
```

What is the difference between HttpServlet and GenericServlet?

Both these classes are abstract but:

GenericServlet	HttpServlet
A GenericServlet has a service() method to handle requests.	The HttpServlet extends GenericServlet and adds support for HTTP protocol based methods like doGet(), doPost(), doHead() etc. All client requests are handled through the **service**() method. The service method dispatches the request to an appropriate method like **doGet**(), **doPost**() etc to handle that request. HttpServlet also has methods like doHead(), doPut(), doOptions(), doDelete(), and doTrace().
Protocol independent. GenericServlet is for servlets that might not use HTTP (for example FTP service).	Protocol dependent (i.e. HTTP).

: What is the difference between doGet () and doPost () or GET and POST?

Prefer using doPost() because it is secured and it can send much more information to the server..

GET or doGet()	POST or doPost()
The request parameters are transmitted as a query string appended to the request. All the parameters get appended to the URL in the address bar. Allows browser bookmarks but not appropriate for transmitting private or sensitive information. http://MyServer/MyServlet?name=paul This is a security risk. In an HTML you can specify as follows: <form name="SSS" method="GET" >	The request parameters are passed with the body of the request. More secured. In HTML you can specify as follows: <form name="SSS" method="**POST**" >
GET was originally intended for static resource retrieval.	POST was intended for form submits where the state of the model and database are expected to change.
GET is not appropriate when large amounts of input data are being transferred. Limited to 1024 characters.	Since it sends information through a socket back to the server and it won't show up in the URL address bar, it can send much more information to the server. Unlike doGet(), it is not restricted to sending only textual data. It can also send binary data such as serialized Java objects.

HttpServletRquest:

It Extends the ServletRequest interface to provide request information for HTTP servlets.

The servlet container creates an HttpServletRequest object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

Frequently used methods:

public java.lang.String getServletPath()

Returns the part of this request's URL that calls the servlet.

public HttpSession getSession()

Returns the current session associated with this request, or if the request does not have a session, creates one.

public HttpSession getSession(boolean create)

Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

java.lang.String	**getContextPath**() Returns the portion of the request URI that indicates the context of the request.
Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
long	**getDateHeader**(java.lang.String name) Returns the value of the specified request header as a long value that represents a Date object.
java.lang.String	**getHeader**(java.lang.String name) Returns the value of the specified request header as a String.
java.util.Enumeration	**getHeaderNames**() Returns an enumeration of all the header names this request contains.
java.util.Enumeration	**getHeaders**(java.lang.String name) Returns all the values of the specified request header as an Enumeration of String objects.
int	**getIntHeader**(java.lang.String name) Returns the value of the specified request header as an int.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
java.lang.String	**getPathInfo**() Returns any extra path information associated with the URL the client sent when it made this request.
java.lang.String	**getPathTranslated**() Returns any extra path information after the servlet name but before the query string, and translates it to a real path.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
java.lang.String	**getRemoteUser**() Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
java.lang.String	**getRequestedSessionId**() Returns the session ID specified by the client.
java.lang.String	**getRequestURI**() Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
java.lang.StringBuffer	**getRequestURL**() Reconstructs the URL the client used to make the request.

HttpServletResponse:

Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies.

The servlet container creates an HttpServletResponse object and passes it as an argument to the servlet's service methods (doGet, doPost, etc).

Frequently used methods:

public void addCookie(Cookie cookie)

Adds the specified cookie to the response.

public void addDateHeader(java.lang.String name, long date)

Adds a response header with the given name and date-value.

public void addHeader(java.lang.String name, java.lang.String value)

Adds a response header with the given name and value.

public java.lang.String encodeRedirectURL(java.lang.String url)

Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.

public java.lang.String encodeURL(java.lang.String url)

Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.

public void sendRedirect(java.lang.String location)

Sends a temporary redirect response to the client using the specified redirect location URL.

public void setDateHeader(java.lang.String name, long date)

Sets a response header with the given name and date-value.

public void setHeader(java.lang.String name, java.lang.String value)

Sets a response header with the given name and value.

public void setStatus(int sc)

Sets the status code for this response.

ServletConfig Interface:

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Below are the methods available in ServletConfig interface.

Method Summary	
java.lang.String	**getInitParameter**(java.lang.String name) Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
java.util.Enumeration	**getInitParameterNames**() Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the servlet has no initialization parameters.
ServletContext	**getServletContext**() Returns a reference to the ServletContext in which the caller is executing.
java.lang.String	**getServletName**() Returns the name of this servlet instance.

Below is the example to add parameters in web.xml file

```
<web-app>
  <servlet>
    <servlet-name>addServlet</servlet-name>
    <servlet-class>AddServlet</servlet-class>
    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
  </servlet>
</web-app>
```

Add below code in service() method for reading the init parameters:

```
ServletConfig config=getServletConfig();
String driver=config.getInitParameter("parametername");
```

If you want to read all init parameters, use below method.

```
Enumeration<String> e=config.getInitParameterNames();
 while(e.hasMoreElements()){
    String name =e.nextElement();
    String value = config.getInitParameter(name));
}
```

ServletContext:

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the <context-param> element.

If you want share common parater meters to across the application, add those parametres in web.xml

```
<web-app>
<servlet>
<servlet-name>servletOne</servlet-name>
<servlet-class>com.test.servlet.ServletOne</servlet-class>
</servlet>

<context-param>
<param-name>parameterName</param-name>
<param-value>parametervalue</param-value>
</context-param>

<servlet-mapping>
<servlet-name>servletOne</servlet-name>
<url-pattern>/urlstring</url-pattern>
</servlet-mapping>
</web-app>
```

If you want to read init parameters form web.xml, add below code in service method.
ServletContext context=getServletContext();
String value=context.getInitParameter("parameterName");

Frequently used methods of ServletContext Interface:

java.lang.Object	**getAttribute**(java.lang.String name) Returns the servlet container attribute with the given name, or null if there is no attribute by that name.
java.util.Enumeration	**getAttributeNames**() Returns an Enumeration containing the attribute names available within this servlet context.
ServletContext	**getContext**(java.lang.String uripath) Returns a ServletContext object that corresponds to a specified URL on the server.
java.lang.String	**getInitParameter**(java.lang.String name) Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist.
java.util.Enumeration	**getInitParameterNames**() Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.
RequestDispatcher	**getNamedDispatcher**(java.lang.String name) Returns a RequestDispatcher object that acts as a wrapper for the named servlet.
java.lang.String	**getRealPath**(java.lang.String path) Returns a String containing the real path for a given virtual path.
RequestDispatcher	**getRequestDispatcher**(java.lang.String path) Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.
java.net.URL	**getResource**(java.lang.String path) Returns a URL to the resource that is mapped to a specified path.
java.io.InputStream	**getResourceAsStream**(java.lang.String path) Returns the resource located at the named path as an InputStream object.
java.util.Set	**getResourcePaths**(java.lang.String path) Returns a directory-like listing of all the paths to resources within the web application whose longest sub-path matches the supplied path argument.
java.lang.String	**getServerInfo**() Returns the name and version of the servlet container on which the servlet is running.
void	**removeAttribute**(java.lang.String name) Removes the attribute with the given name from the servlet context.
void	**setAttribute**(java.lang.String name, java.lang.Object object) Binds an object to a given attribute name in this servlet context.

Servlet Collaboration:

RequestDispatcher Interface:

Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server. The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server resource located at a particular path or given by a particular name.

getRequestDispatcher():

public RequestDispatcher getRequestDispatcher(java.lang.String path)

Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path. A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

The pathname must begin with a "/" and is interpreted as relative to the current context root. Use getContext to obtain a RequestDispatcher for resources in foreign contexts. This method returns null if the ServletContext cannot return a RequestDispatcher.

getNamedDispatcher():

public RequestDispatcher getNamedDispatcher(java.lang.String name)

Returns a RequestDispatcher object that acts as a wrapper for the named servlet.

Servlets (and JSP pages also) may be given names via server administration or via a web application deployment descriptor. A servlet instance can determine its name using ServletConfig.getServletName().

This method returns null if the ServletContext cannot return a RequestDispatcher for any reason.

We can get RequestDispatcher object in different ways.

ServletContext.getRequestDispatcher(java.lang.String);

ServletContext.getNamedDispatcher(java.lang.String);

ServletRequest.getRequestDispatcher(java.lang.String)

RequestDispatcher having two methods.
1. forward()
2. include()

public void forward(ServletRequest request,ServletResponse response) throws ServletException, java.io.IOException

Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server. This method allows one servlet to do preliminary processing of a request and another resource to generate the response.

For a RequestDispatcher obtained via getRequestDispatcher(), the ServletRequest object has its path elements and parameters adjusted to match the path of the target resource.

forward should be called before the response has been committed to the client (before response body output has been flushed). If the response already has been committed, this method throws an IllegalStateException. Uncommitted output in the response buffer is automatically cleared before the forward.

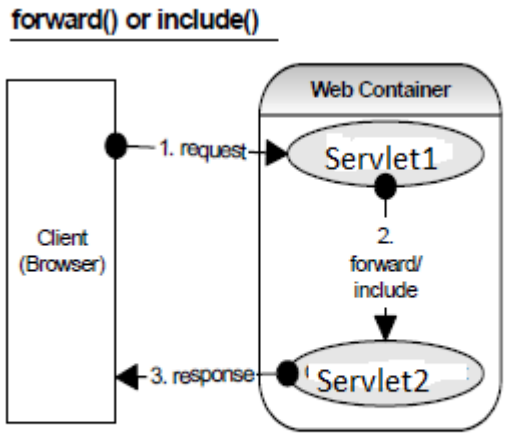
The request and response parameters must be either the same objects as were passed to the calling servlet's service method or be subclasses of the ServletRequestWrapper or ServletResponseWrapper classes that wrap them.

public void include(ServletRequest request, ServletResponse response) throws ServletException,java.io.IOException

Includes the content of a resource (servlet, JSP page, HTML file) in the response. In essence, this method enables programmatic server-side includes.

The ServletResponse object has its path elements and parameters remain unchanged from the caller's. The included servlet cannot change the response status code or set headers; any attempt to make a change is ignored.

The request and response parameters must be either the same objects as were passed to the calling servlet's service method or be subclasses of the ServletRequestWrapper or ServletResponseWrapper classes that wrap them.



What is the difference between the getRequestDispatcher(String path) method of ServletRequest interface and ServletContext interface?

javax.servlet.ServletRequest getRequestDispatcher(String path)

Accepts path parameter of the servlet or JSP to be included or forwarded relative to the request of the calling servlet. If the path begins with a “/” then it is interpreted as relative to current context root.

javax.servlet.ServletContext getRequestDispatcher(String path)

Does not accept relative paths and all path must start with a “/” and are interpreted as relative to current context root.

RequestDispatcher Example:

Create web project:

1. open eclipse
2. go to File --> New --> select "Dynamic Web Project"
3. Fill below values

   Project name: RegistrationApp

   Dynamic web module version: 3.0 0r later version

   click Next

   select Generate web.xml deployment descriptor
4. Create person.html file under "WebContent" folder

```
<!DOCTYPE html>
<html><body>
 <form action="registration" method="post">
 Fill below details for Applying the voter card : <br>
 <table>
    <tr>   <td>First Name                                  : </td>
           <td><input type="text" name="fname" id ="fname"></td>   </tr>
                  <tr>
                         <td>Last Name                                          : </td>
                         <td><input type="text" name="lname" id ="lname"> </td>
                  </tr>
                  <tr>
                         <td>Age                                                                  : </td>
                         <td><input type="text" name="age" id ="age"> </td>
                  </tr>
                  <tr>
                         <td>Address                                          : </td>
                         <td><input type="text" name="address" id ="address"> </td> </tr>
 </table>
 <input type="submit" value="Submit">
 </form>
</body>
</html>
```

5. Create RegistrationServlet.java class under src folder

```
@WebServlet("/registration")
public class RegistrationServlet extends HttpServlet {
		private static final long serialVersionUID = 1L;

		protected void doPost(HttpServletRequest request, HttpServletResponse response)
					throws ServletException, IOException {

			int age = Integer.parseInt(request.getParameter("age"));
			if(age > 18) {
					RequestDispatcher rd = request.getRequestDispatcher("/confirmation");
					rd.forward(request, response);
			} else {
					PrintWriter out = response.getWriter();
					out.println("You are not eligibal for voter card because you age is less than 18 years.");
			}
		}

}
```

6. Create ConfirmationServlet.java class under src folder.

```
@WebServlet("/confirmation")
public class ConfirmationServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                out.println("Thank you for applaying voter card and Your registration details are:");
                out.println("Name:"+request.getParameter("fname") + " "+request.getParameter("lname"));
                out.println("Age:"+request.getParameter("age"));
                out.println("Address:"+request.getParameter("address"));
                out.println("Reference Number:"+new Random().nextInt());
        }
}
```

Note: launch the browser and run the below url
http://localhost:8080/ RegistrationApp

sendRedirect method:

public void sendRedirect(String location) throws java.io.IOException

Sends a temporary redirect response to the client using the specified redirect location URL. This method can accept relative URLs; the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is relative without a leading '/' the container interprets it as relative to the current request URI. If the location is relative with a leading '/' the container interprets it as relative to the servlet container root.

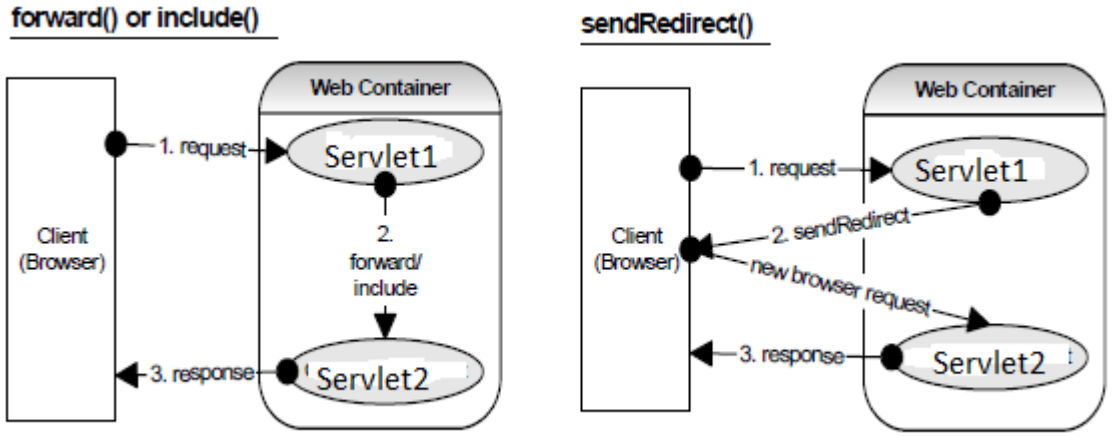
If the response has already been committed, this method throws an IllegalStateException. After using this method, the response should be considered to be committed and should not be written to.

Example:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.sendRedirect(“registration");
        //response.sendRedirect(“person.html");
        //response.sendRedirect(“www.google.com");
}
```

What is the difference between forwarding a request and redirecting a request?

sendRedirect()	Forward()
Sends a header back to the browser, which contains the name of the resource to be redirected to. The browser will make a **fresh request from this header information**. Need to provide absolute URL path.	Forward action takes place within the server **without the knowledge of the browser.** Accepts relative path to the servlet or context root.
Has an overhead of extra remote trip but has the advantage of being able to refer to any resource on the same or different domain and also allows book marking of the page.	No extra network trip.



Session Management:

http protocol is a stateless request/response based protocol.
HTTP is a stateless protocol, so, how do you maintain state? How do you store user data between requests?

A session identifies the requests that originate from the same browser during the period of conversation.

All the servlets can share the same session.

We can obtain session in 4 ways.

1. Cookies
2. Url re-writing
3. Hidden form fields
4. HttpSession

Cookies session tracking:

Cookie is a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server. A cookie's value can uniquely identify a client, so cookies are commonly used for session management.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number. Some Web browsers have bugs in how they handle the optional attributes, so use them sparingly to improve the interoperability of your servlets.

The servlet sends cookies to the browser by using the **HttpServletResponse.addCookie(javax.servlet.http.Cookie)** method, which adds fields to HTTP response headers to send cookies to the browser, one at a time. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.

The browser returns cookies to the servlet by adding fields to HTTP request headers. Cookies can be retrieved from a request by using the **HttpServletRequest.getCookies()** method. Several cookies might have the same name but different path attributes.

Create Cookie :

```
Cookie cookie=new Cookie("name","value");
response.addCookie(cookie);
```

Get cookies:

```
Cookie cookies[]=request.getCookies();
for(Cookie c:cookies){
   System.out.println("cookie name:"+c.getName() +"cookie value:"+c.getValue());
}
```

Delete cookies:

```
Cookie cookie =new Cookie("name",””);
ck.setMaxAge(0);
response.addCookie(cookie);
```

Points to remember with respect to cookies session tracking:

- There is a limit for cookie size.
- The browser may turn off cookies.
- The performance is moderate.
- The benefit of the cookies is that state information can be stored regardless of which server the client talks to and even if all servers go down. Also, if required, state information can be retained across sessions.

URL re-writing:

URL re-writing will append the state information as a query string to the URL.

This should not be used to maintain private or sensitive information.

We can send parameter name/value pairs using the below format:

http://Serveraddress:8080/myServlet?firstname=Tom&lastname=jhon

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

Points to remember:

- There is a limit on the size of the session data.
- Should not be used for sensitive or private information.
- The performance is moderate.

Hidden form fields:

Hidden Fields on the pages can maintain state and they are not visible on the browser. The server treats both hidden and non-hidden fields the same way.

Example:

<input type="hidden" name="firstname” value="Peter">

<input type="hidden" name=“lastname" value="Jhon">

The disadvantage of hidden fields is that they may expose sensitive or private information to others.

Points to remember:

- There is no limit on size of the session data.
- May expose sensitive or private information to others (So not good for sensitive information).
- The performance is moderate.

HttpSession:

Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server.

The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times.

How to create HttpSession object?

HttpSession session = request.getSession(true);

getSession(true): This method will check whether there is already a session exists for the user. If a session

exists, it returns that session object. If a session does not already exist then it creates a new session for the user.

HttpSession session = request.getSession(false);

getSession(false): This method will check whether there is already a session exists for the user. If a session exists, it returns that session object. If a session does not already exist then it returns null.

setAttribute():

public void setAttribute(String name,Object value)

Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.

getAttribute():

public java.lang.Object getAttribute(String name)

Returns the object bound with the specified name in this session, or null if no object is bound under the name.

Points to remember:

- There is no limit on the size of the session data kept.
- The performance is good.
- HTTP Sessions are the recommended approach.

Note: When using HttpSession mechanism you need to take care of the following points:

- Remove session explicitly when you no longer require it.
- Set the session timeout value.
- Your application server may serialize session objects after crossing a certain memory limit. This is expensive and affects performance. So decide carefully what you want to store in a session.

How to invalidate session object?

If a session is no longer required e.g. user has logged out, etc then it can be invalidated using invalidate() method.

```
session.invalidate();
```

How to set the session time out in servlet?

1. Set the session inactivity lease period on a per session basis

session.setMaxInactiveInterval(300); //resets inactivity period for this session as 5 minutes

2. set session timeout in web.xml file

```
<session-config>
  <session-timeout>5</session-timeout>
</session-config>
```

State mechanism	Description
HttpSession	▪ There is no limit on the size of the session data kept. ▪ The performance is good. ▪ This is the preferred way of maintaining state. If we use the HTTP session with the application server's persistence mechanism (server converts the session object into BLOB type and stores it in the Database) then the performance will be moderate to poor. Note: When using HttpSession mechanism you need to take care of the following points: ▪ Remove session explicitly when you no longer require it. ▪ Set the session timeout value. ▪ Your application server may serialize session objects after crossing a certain memory limit. This is expensive and affects performance. So decide carefully what you want to store in a session.
Hidden fields	▪ There is no limit on size of the session data. ▪ May expose sensitive or private information to others (So not good for sensitive information). ▪ The performance is moderate.
URL rewriting	▪ There is a limit on the size of the session data. ▪ Should not be used for sensitive or private information. ▪ The performance is moderate.
Cookies	▪ There is a limit for cookie size. ▪ The browser may turn off cookies. ▪ The performance is moderate. The benefit of the cookies is that state information can be stored regardless of which server the client talks to and even if all servers go down. Also, if required, state information can be retained across sessions.

Attributes in Servlet:

What are the different scopes or places where a servlet can save data for its processing?

Request Scope:

Data saved in a request-scope goes out of scope once a response has been sent back to the client (i.e. when the request is completed).

save and get request-scoped value

```
request.setAttribute("calc-value", new Float(7.0));
request.getAttribute("calc-value");
```

Session Scope:

Data saved in a session-scope is available across multiple requests. Data saved in the session is destroyed when the session is destroyed (not when a request completes but spans several requests).

save and get session-scoped value

```
HttpSession session = request.getSession(false);
If(session != null) {
session.setAttribute("id", "DX12345");
value = session.getAttribute("id");
}
```

Application scope:

Data saved in a ServletContext scope is shared by all servlets and JSPs in the context. The data stored in the servlet context is destroyed when the servlet context is destroyed.

save and get an application-scoped value

```
getServletContext().setAttribute("application-value", "shopping-app");
value = getServletContext().getAttribute("application-value");
```

Filters Interface:

- A filter is an object that is invoked at the preprocessing and postprocessing of a request.
- A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both.
- Filters perform filtering in the doFilter method. Every Filter has access to a FilterConfig object from which it can obtain its initialization parameters, a reference to the ServletContext which it can use, for example, to load resources needed for filtering tasks.
- Filters are configured in the deployment descriptor of a web application

Examples:

1) Authentication Filters
2) Logging and Auditing Filters
3) Image conversion Filters
4) Data compression Filters
5) Encryption Filters
6) Tokenizing Filters
7) Filters that trigger resource access events
8) XSL/T filters
9) Mime-type chain Filter

Init method:

public void init(FilterConfig filterConfig) throws ServletException

Called by the web container to indicate to a filter that it is being placed into service. The servlet container calls the init method exactly once after instantiating the filter. The init method must complete successfully before the filter is asked to do any filtering work.

doFilter method:

public void doFilter(ServletRequest request,ServletResponse response, FilterChain chain)

throws java.io.IOException,ServletException

The doFilter method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain. The FilterChain passed in to this method allows the Filter to pass on the request and response to the next entity in the chain.

A typical implementation of this method would follow the following pattern:

1. Examine the request

2. Optionally wrap the request object with a custom implementation to filter content or headers for input filtering

3. Optionally wrap the response object with a custom implementation to filter content or headers for output filtering

4. a) Either invoke the next entity in the chain using the FilterChain object (chain.doFilter()),

4. b) or not pass on the request/response pair to the next entity in the filter chain to block the request processing

5. Directly set headers on the response after invocation of the next entity in the filter chain.

destroy method:

public void destroy()

Called by the web container to indicate to a filter that it is being taken out of service. This method is only called once all threads within the filter's doFilter method have exited or after a timeout period has passed. After the web container calls this method, it will not call the doFilter method again on this instance of the filter.

This method gives the filter an opportunity to clean up any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the filter's current state in memory.

Filter example:

1.login.html

```
<!DOCTYPE html>
<html>
<body>
<form action="login">
	Login Details:
				User name  : <input type="text" name="username" id="username"><br>
				Password : <input type="password" name="password" id="password"><br>
				<input type="submit" value="LogIn">
		</form>
</body>
</html>
```

2.LoginServlet.java

```
public class LoginServlet extends HttpServlet {
		public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
				String userName = request.getParameter("username");
				PrintWriter out = response.getWriter();
				out.println("Welcome to "+userName);
		}
}
```

3.LoginFilter.java

```
public class LoginFilter implements Filter{
		public void init(FilterConfig filterConfig)  throws ServletException {
			System.out.println("init method invoked..");
		}
		@Override
		public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
					throws IOException, ServletException {
			response.setContentType("text/html");
			PrintWriter out = response.getWriter();
			String userName = request.getParameter("username");
			if(userName.endsWith("rao")) {
				out.println("Sorry you are not valid user.");
				out.println("<br><br>");
				out.println("<a href='/FilterExample/login.html'>Login</a>");
				return;
			}
			filterChain.doFilter(request, response);
			out.println("<br><br>");
			out.println("Have a nice day");
		}
		public void destroy() {
			System.out.println("destroy method invoked..");
		}}
```

4. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns="http://xmlns.jcp.org/xml/ns/javaee"
                xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
                id="WebApp_ID" version="4.0">
                <display-name>FilterExample</display-name>
                <welcome-file-list>
                                <welcome-file>login.html</welcome-file>
                </welcome-file-list>
                <servlet>
                                <servlet-name>loginServlet</servlet-name>
                                <servlet-class>com.test.servlet.LoginServlet</servlet-class>
                </servlet>
                <servlet-mapping>
                                <servlet-name>loginServlet</servlet-name>
                                <url-pattern>/login</url-pattern>
                </servlet-mapping>
                <filter>
                                <filter-name>loginFilter</filter-name>
                                <filter-class>com.test.servlet.LoginFilter</filter-class>
                </filter>
                <filter-mapping>
                                <filter-name>loginFilter</filter-name>
                                <url-pattern>/login</url-pattern>
                </filter-mapping>
</web-app>
```

User below link for accessing the application:

http://localhost:8080/FilterExample/

Events and Listeners:

The servlet specification includes the capability to track key events in your Web applications through event listeners. This functionality allows more efficient resource management and automated processing based on event status.

(Changing the state of an object is known as an event.)

There are two levels of servlet events:

1. Servlet context-level (application-level) event:

This event involves resources or state held at the level of the application servlet context object.

2. Session-level event:

This event involves resources or state associated with the series of requests from a single user session; that is, associated with the HTTP session object.

Each of these two levels has two event categories:

Lifecycle changes

Attribute changes

Event Listener Categories and Interfaces:

Event Category	Event Descriptions	Java Interface
Servlet context lifecycle changes	Servlet context creation, at which point the first request can be serviced	javax.servlet. ServletContextListener
	Imminent shutdown of the servlet context	
Servlet context attribute changes	Addition of servlet context attributes	javax.servlet. ServletContextAttributeListener
	Removal of servlet context attributes	
	Replacement of servlet context attributes	
Session lifecycle changes	Session creation	javax.servlet.http. HttpSessionListener
	Session invalidation	
	Session timeout	
Session attribute changes	Addition of session attributes	javax.servlet.http. HttpSessionAttributeListener
	Removal of session attributes	
	Replacement of session attributes	

HttpSessionListener:

It extends java.util.EventListener

Implementations of this interface are notified of changes to the list of active sessions in a web application. To receive notification events, the implementation class must be configured in the deployment descriptor for the web application.

Available Methods:

sessionCreated method:

public void sessionCreated(HttpSessionEvent se)

Notification that a session was created.

sessionDestroyed method:

public void sessionDestroyed(HttpSessionEvent se)

Notification that a session is about to be invalidated.

Example:

Create web project and provide name as “ListenerExample” and add below components respective folders.

1.postquery.html

```
<!DOCTYPE html>
<html>
<body>
        <form action="postQuery">
    Post your query:
                Query  : <input type="text" name="query" id="query"><br>
                <input type="submit" value="Post Query">
        </form>
</body>
</html>
```

2.SubmitQueryServlet.java

```
@WebServlet("/postQuery")
public class SubmitQueryServlet 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();
			HttpSession session = request.getSession();
			session.setMaxInactiveInterval(60);
			out.println("<html>");
			out.println("<body>");
			out.println("We are received you query. Thanks for submmitting the query.");
			out.println("<h2>Number of Users Online : " + UserCountListener.getNumberOfUsersOnline() + "</h2>");
			out.println("</body>");
			out.println("</html>");
		}

}
```

3. UserCountListener.java

```
@WebListener
public class UserCountListener implements HttpSessionListener {

		private static int numberOfUsersOnline;

		public UserCountListener() {
			numberOfUsersOnline = 0;
		}
		public static int getNumberOfUsersOnline() {
			return numberOfUsersOnline;
		}
		public void sessionCreated(HttpSessionEvent se) {
			System.out.println("Session created and Id is: " + se.getSession().getId());
			synchronized (this) {
				numberOfUsersOnline++;
			}
		}
		public void sessionDestroyed(HttpSessionEvent se) {
			System.out.println("Session destroyed for Id : " + se.getSession().getId());
			synchronized (this) {
				numberOfUsersOnline--;
			}
		}
}
```

4. Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID"
version="4.0">
  <display-name>FilterExample</display-name>
  <welcome-file-list>
    <welcome-file>postquery.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Access the application using below link.
http://localhost:8080/ListenerExample