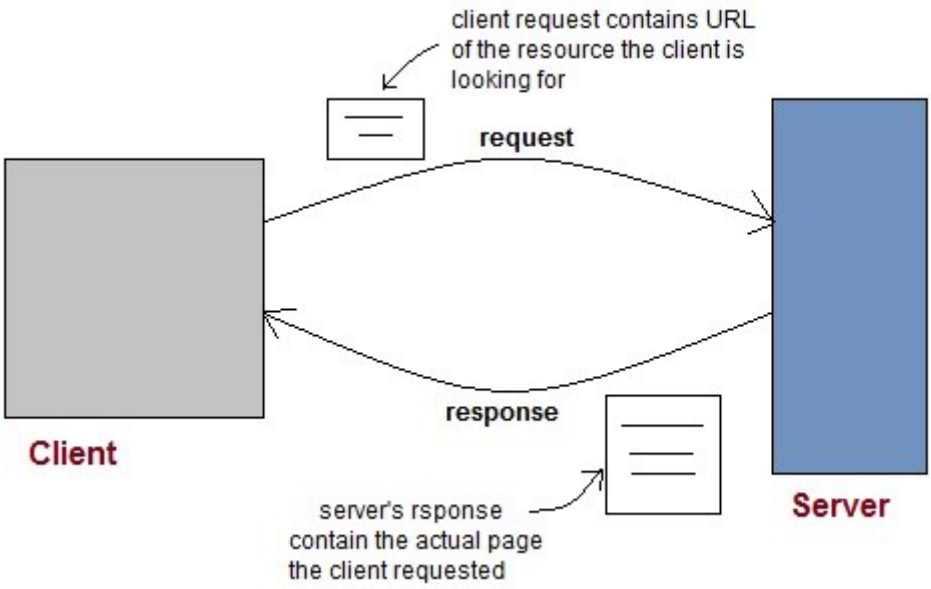
CHAPTER 3
SERVLET

1. INTRODUCTION

1.1. Introduction to web Processing:

Web contains billions client and server which connected though network. Web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.



1.2. Web Server, Application Server and Web Client:

Web Server: Web Server is a software that can process the client request and send the response back to the client. For example, Apache is one of the most widely used web server. Web Server runs on some physical machine and listens to client request on specific port.

Application Server: Most of the application servers have Web Server as integral part of them, that means App Server can do whatever Web Server is capable of. Additionally App Server have components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services etc. e.g. WebLogic

Web Client: A web client is a software that helps in communicating with the server. Some of the most widely used web clients are Firefox, Google Chrome, Safari etc. When we request something from server (through URL), web client takes care of creating a request and sending it to server and then parsing the server response and present it to the user.

1.3. HTTP:

- HTTP stands for hypertext tranfer protocol.
- It is used to send files (text, graphics, images, sounds, video and other multimedia files) on the web.
- It's a stateless, application-layer protocol for communicating between distributed systems.
- The current version of the protocol is **HTTP/1.1.**
- The default port number for HTTP is 80.

1. HTTP Methods and Descriptions :

Method Name	Description
OPTIONS	Request for communication options that are available on the request/response chain.
GET	Request to retrieve information from server using a given URI.
HEAD	Identical to GET except that it does not return a message-body, only the headers and status line.
POST	Request for server to accept the entity enclosed in the body of HTTP method.
DELETE	Request for the Server to delete the resource.
CONNECT	Reserved for use with a proxy that can switch to being a tunnel.
PUT	This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content.

2. Web Application:

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

1.4. Common Gateway Interface:

CGI(Common Gateway Interface) which is used before the Servlet. It is used to create web application.

1. Working of CGI:

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web servers run the CGI program in seperate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the web server, which wraps the response in an HTTP response and send it back to the web browser.

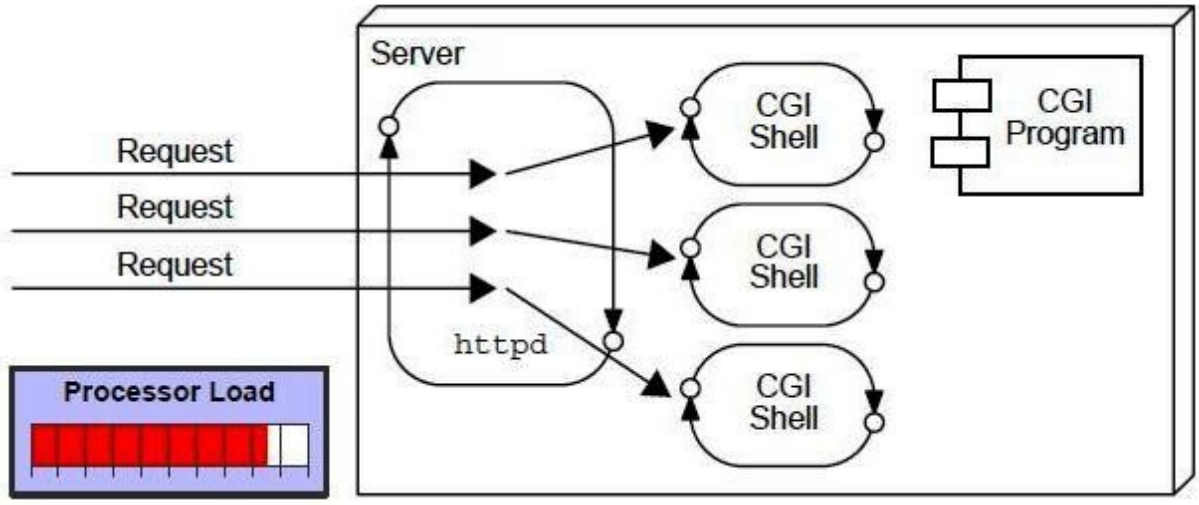


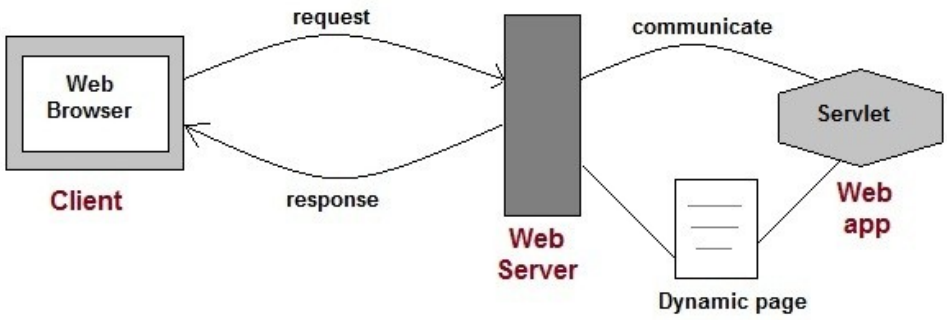
Fig. Working of CGI

2. Drawbacks of CGI:

- High resposne time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

1.5. Introduction to Servlet:

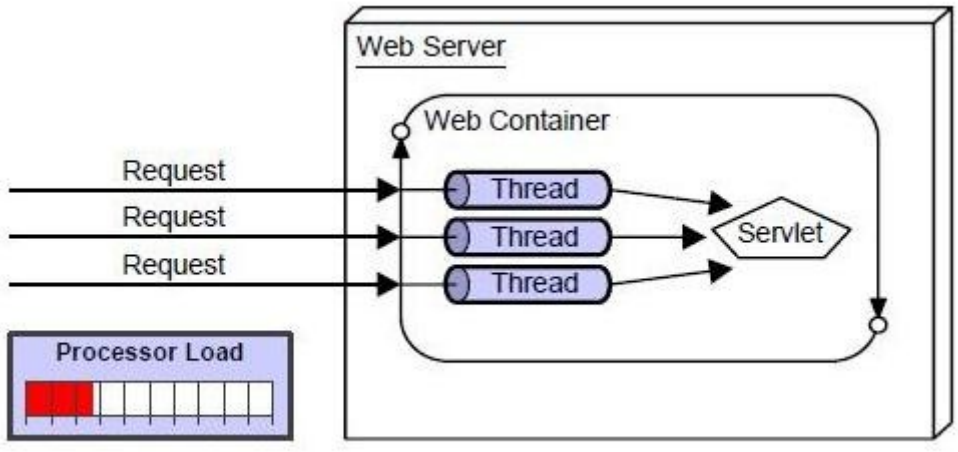
Servlets are modules that run inside request/response-oriented servers, such as Java-enabled web servers. Functionally they operate in a very similar way to CGI scripts, however, being Java based they are more platform independent.



Servlet

Servlet Technology is used to create web applications. Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

1. Working of Servlet:



After receiving any request from client first web container checks whether instsnce for requested servlet is existed or not. If the servlet instance is exist then container pass that request to instance. If instance is not existed container laods the servlet file and creates the instance of the requested servlet.

2. Difference between CGI and Servlets:

CGI	Servlet
Platform Dependant	platform independant
every request is processed by own heavy process	every request is processed by thread which is lightweight
more vulnerable to attacks than servlets	Les vulnerable to attacks as it in java language
Not much stronger as Servlets	Stronger than CGI because of Strong memory management and exception handling
CGI does not provide sharing property.	Servlets can share data among each other.
CGI cannot read and set HTTP headers, handle cookies, tracking sessions	Servlets can read and set HTTP headers, handle cookies, tracking sessions.

3. Applications of Servlet:

- Processing data POSTed over HTTPS using an HTML form, including purchase order or credit card data: A servlet like this could be part of an order-entry and processing system,

working with product and inventory databases, and an on-line payment system.

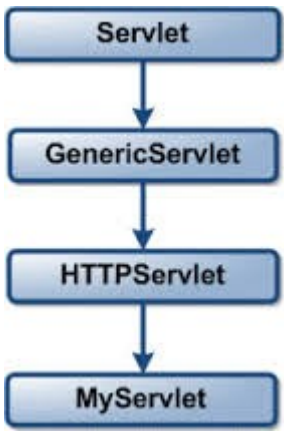
- Allowing collaboration between people: A servlet can handle multiple requests concurrently; they can synchronize requests to support systems such as on-line conferencing.
- Forwarding requests: Servlets can forward requests to other servers and servlets. This allows them to be used to balance load among several servers that mirror the same content. It also allows them to be used to partition a single logical service over several servers, according to task type or organizational boundaries.

4. Advantages of using Servlet:

- Servlet technology was introduced to overcome the shortcomings of CGI technology.
- It provides better performance that CGI in terms of processing time, memory utilization because servlets uses benefits of multithreading and for each request a new thread is created, that is faster than loading creating new Object for each request with CGI.
- Servlets are platform independent, the web application developed with Servlet can be run on any standard web container such as Tomcat, JBoss, Glassfish servers and on operating systems such as Windows, Linux, Unix, Solaris, Mac etc.
- Servlets are robust because container takes care of life cycle of servlet and we don't need to worry about memory leaks, security, garbage collection etc.
- Servlets are maintainable.

1.6. Servlet Architecture:

All servlets implement Servlet interface, either directly or, more commonly, by extending a class HttpServlet. HTTPServlet indirectly implements Servlet interface.



The Servlet interface provides the following methods that manage the servlet and its communications with clients.

init, service, getServletConfig, getServletInfo, destroy

When a servlet accepts a service call from a client, it receives two objects, ServletRequest and ServletResponse. The ServletRequest class encapsulates the communication from the client to the server, while the ServletResponse class encapsulates the communication from the servlet back to the client.
The ServletRequest interface allows the servlet access to information such as the names of the parameters passed in by the client, the protocol (scheme) being used by the client, and the names of the remote host that made the request and the server that received it. It also provides the servlet with access to the input stream, ServletInputStream, through which the servlet gets data from clients that are using application protocols such as the HTTP POST and PUT methods. Subclasses of ServletRequest allow the servlet to retrieve more protocol-specific data. For example, HttpServletRequest contains methods for accessing HTTP-specific header information.
The ServletResponse interface gives the servlet methods for replying to the client. It allows the servlet to set the content length and mime type of the reply, and provides an output stream, ServletOutputStream, and a Writer through which the servlet can send the reply data. Subclasses of ServletResponse give the servlet more protocol-specific capabilities. For example, HttpServletResponse contains methods that allow the servlet to manipulate HTTP-specific header information.

HTTP servlets have some additional objects that provide session-tracking capabilities. The servlet writer can use these APIs to maintain state between the servlet and the client that persists across multiple connections during some time period.

1.7. Servlet Life Cycle:

1. **Loading Servlet Class :**

- A Servlet class is loaded when first request for the servlet is received by the Web Container.

2. **Servlet instance creation :**

- After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.

3. **Call to the init() method :**

- init() method is called by the Web Container on servlet instance to initialize the servlet.
- Syntax: public void init(ServletConfig config) throws ServletException.
- The init method is designed to be called only once.
- It is called when the servlet is first created, and not called again for each user request.

4. **Call to the service() method :**

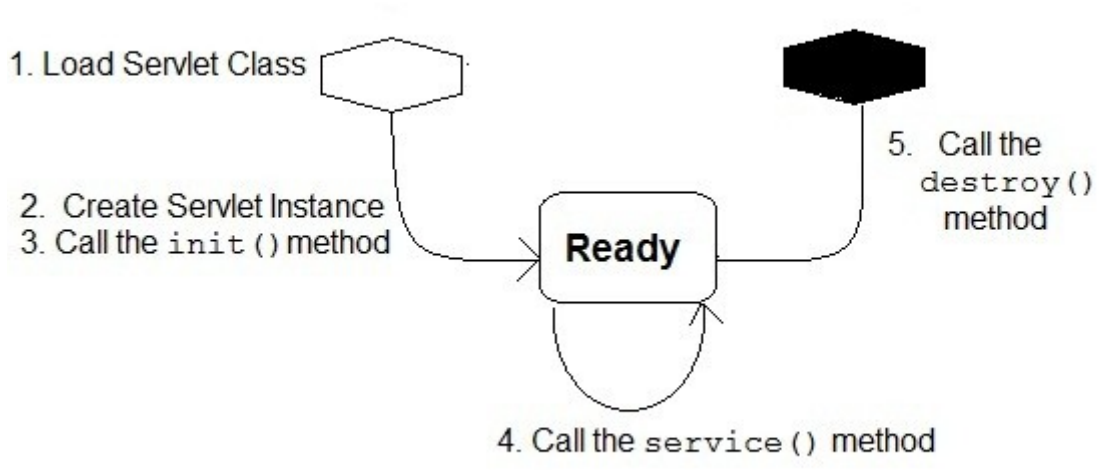
- The containers call the service() method each time the request for servlet is received.

- The service() method will then call the doGet, doPost, doPut, doDelete, etc. methods as appropriate based on the type of the HTTP request.
- Syntax: public void **service**(ServletRequest request, ServletResponse response) throws ServletException, IOException
- The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So we override either doGet() or doPost() depending on what type of request we receive from the client.

5. **Call to destroy() method:**

- The Web Container calls the destroy() method before removing servlet instance for cleanup activity.
- Syntax:
```
public void destroy() {
        // Finalization code...
}
```

- It is called only once at the end of life cycle
- Then it is marked for garbage collection



1. HttpServlet Methods:

doGet method:

- It is used to serve the request of user.
- It will get called implicitly if no method is mentioned.
- The service method calles this method according to user requirement.
- Syntax:
  public void doGet(HttpServletRequest request, HttpServletResponse response)throws

```
ServletException, IOException {
                    // Servlet code
}
```

doPost method:

- It is also used to serve the request of user.
- It has to be called explicitly whenever user needs to pass secure contents.
- The service method calles this method according to user requirement.
- Syntax:

```
public void doPost(HttpServletRequest request,  HttpServletResponse response) throws ServletException, IOException {
                    // Servlet code
}
```

doPut method:

- The doPut() method handles requests sent using the HTTP PUT method.
- The PUT method allows a client to store information on the server.
- For an example, you can use it to post an image file to the server
- Syntax:

```
public void doPut(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                    // Servlet code
}
```

doDelete method:

- It is called by the server (via the `service` method) to allow a servlet to handle a DELETE request.
- The DELETE operation allows a client to remove a document or Web page from the server.

API : https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServlet.html

2. Diffrence between GET and POST methods:

Feature	GET	POST
Sending of data	Client data is appended to URL and sent	Client data is sent separately
Storing in browser history	As data is appended, the client data is stored in browser history	As data is sent seaprately, the client data is not stored in browser history
Bookmark	The URL with client data can be bookmarked. Thereby, later without filling the HTML form, the same data can be sent to server	Not possible to bookmark
Limitation of data sent	Limited to 2048 characters (browser dependent)	Unlimited data

Hacking easiness	Easy to hack the data as data is stored in browser history	Difficult to hack
Type of data sent	Only ASCII data can be sent	Any type of data can be sent including binary data
Data secrecy	Data is not secret as other people can see the data in browser history	Data is secret as not stored in history
When to be used	Prefer when data sent is not secret. Do not use for passwords etc.	Prefer for critical and sensitive data like passwords etc.
Cache	Can be caught	Cannot be caught
Default	If not mentioned, GET is assumed as default	Should be mentioned explicitly
Performance	Relatively faster as data is appeneded to URL	A separate message body is to be created

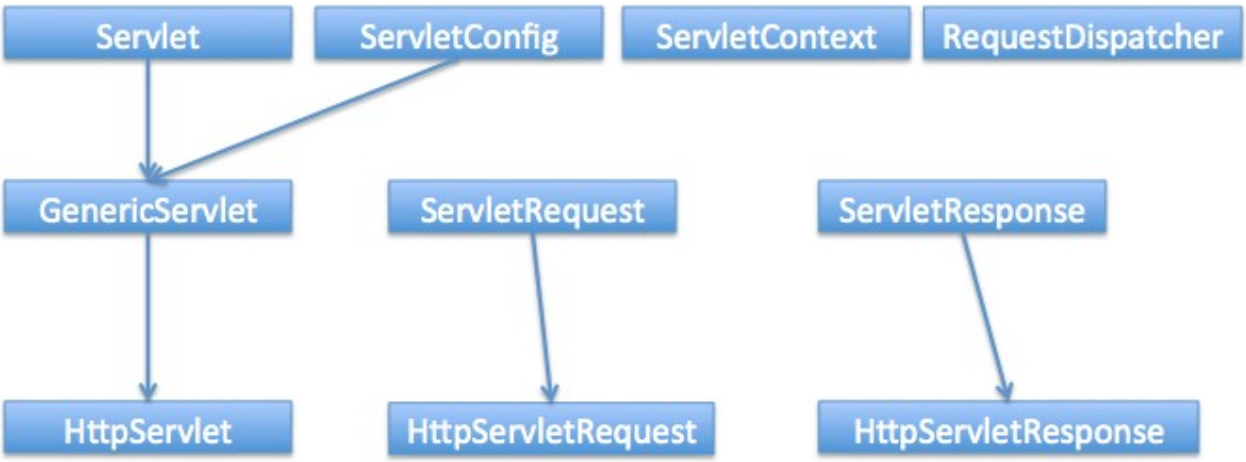
2. SERVLET API

2.1. Servlet API History:

Servlet API version	Released	Platform	Important Changes
Servlet 3.1	May 2013	JavaEE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)
Servlet 3.0	December 2009	JavaEE 6, JavaSE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Servlet 2.5	September 2005	JavaEE 5, JavaSE 5	Requires JavaSE 5, supports annotation
Servlet 2.4	November 2003	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Servlet 2.3	August 2001	J2EE 1.3, J2SE 1.2	Addition of Filter
Servlet 2.2	August 1999	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Servlet 2.1	November 1998	Unspecified	First official specification, added RequestDispatcher, ServletContext

2.2. Servlet API Hierarchy:

javax.servlet.Servlet is the base interface of Servlet API. There are some other interfaces and classes that we should be aware of when working with Servlets. Also with Servlet 3.0 specs, servlet API introduced use of annotations rather than having all the servlet configuration in deployment descriptor. In this section, we will look into important Servlet API interfaces, classes and annotations. The below diagram shows servlet API hierarchy.



Servlet

1. Servlet Interface:

javax.servlet.Servlet is the base interface of **Java Servlet API**. Servlet interface declares the life cycle methods of servlet. All the servlet classes are required to implement this interface. The methods declared in this interface are:

Methods	Description
public abstract void init(ServletConfig paramServletConfig) throws ServletException	This is the very important method that is invoked by servlet container to initialized the servlet and ServletConfig parameters. The servlet is not ready to process client request until unless **init()** method is finished executing. This method is called only once in servlet lifecycle and make Servlet class different from normal java objects. We can extend this method in our servlet classes to initialize resources such as DB Connection, Socket connection etc.
public abstract ServletConfig getServletConfig()	This method returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. We can use this method to get the init parameters of servlet defines in deployment descriptor (web.xml) or through annotation in Servlet 3. We will look into ServletConfig interface later on.
public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException	This method is responsible for processing the client request. Whenever servlet container receives any request, it creates a new thread and execute the service() method by passing request and response as argument. Servlets usually run in multi-threaded environment, so it's developer responsibility to keep shared resources thread-safe using **synchronization**.
public abstract String getServletInfo()	This method returns string containing information about the servlet, such as its author, version, and copyright. The string returned should be plain text and can't have markups.
public abstract void destroy()	This method can be called only once in servlet life cycle and used to close any open resources. This is like finalize method of a java class.

Link for API: https://docs.oracle.com/cd/E17802_01/products/products/servlet/2.5/docs/servlet-2_5-mr2/javax/servlet/Servlet.html

2. ServletConfig Interface:

javax.servlet.ServletConfig is used to pass configuration information to Servlet. Every servlet has

it's own ServletConfig object and servlet container is responsible for instantiating this object. We can provide servlet init parameters in **web.xml** file or through use of WebInitParam annotation. We can use **getServletConfig()** method to get the ServletConfig object of the servlet.

The important methods of ServletConfig interface are:

Method	Description
public abstract String getInitParameter(String paramString)	This method can be used to get the specific init parameter value by name. If parameter is not present with the name, it returns null.
public abstract String getInitParameterNames(String paramString)	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.
public abstract ServletContext getServletContext()	This method returns the ServletContext object for the servlet.
public abstract ServletContext getServletName()	Returns the name of this servlet instance. The name may be provided via server administration, assigned in the web application deployment descriptor, or for an unregistered (and thus unnamed) servlet instance it will be the servlet's class name.

Link for API: https://tomcat.apache.org/tomcat-4.0-doc/servletapi/javax/servlet/ServletConfig.html

3. ServletContext interface:

javax.servlet.ServletContext interface provides access to web application variables to the servlet. The ServletContext is unique object and available to all the servlets in the web application. When we want some init parameters to be available to multiple or all of the servlets in the web application, we can use ServletContext object and define parameters in web.xml using **<context-param>** element. We can get the ServletContext object via the getServletContext() method of ServletConfig. Servlet engines may also provide context objects that are unique to a group of servlets and which is tied to a specific portion of the URL path namespace of the host.

Some of the important methods of ServletContext are:

Methods	Description
public abstract Object getAttribute(String name)	Returns the servlet container attribute with the given name, or null if there is no attribute by that name.
public abstract Object getAttributeNames(String name)	Returns an Enumeration containing the attribute names available within this servlet context.
String getInitParameter(String	This method returns the String value for the init parameter

name)	defined with name in web.xml, returns null if parameter name doesn't exist. We can use Enumeration to get enumeration of all the init parameter names.
String getInitParameterNames(String name)	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.
public abstract ServletContext getContext(String uripath)	This method returns ServletContext object for a particular uripath or null if not available or not visible to the servlet.
public abstract URL getResource(String path) throws MalformedURLException	This method return URL object allowing access to any content resource requested. We can access items whether they reside on the local file system, a remote file system, a database, or a remote network site without knowing the specific details of how to obtain the resources.
public abstract InputStream getResourceAsStream(String path)	This method returns an input stream to the given resource path or null if not found.
8. public abstract RequestDispatcher getRequestDispatcher(String urlpath)	This method is mostly used to obtain a reference to another servlet. After obtaining a RequestDispatcher, the servlet programmer forward a request to the target component or include content from it.
public abstract void log(String msg)	This method is used to write given message string to the servlet log file.
public abstract void setAttribute(String paramString, Object paramObject)	This method is used to set the attribute with application scope. The attribute will be accessible to all the other servlets having access to this ServletContext. We can remove an attribute using public abstract void removeAttribute(String paramString) method.
boolean setInitParameter(String paramString1, String paramString2)	Return the object attribute for the given name. We can get enumeration of all the attributes using public abstract Enumeration method.

Link for API: https://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html

Example of Config and Context:

Servlet 1: UserInfo.java

```
public class UserInfo extends HttpServlet {
        private static final long serialVersionUID = 1L;

            /**
```

```
         * @see HttpServlet#HttpServlet()
         */
        public UserInfo() {
            super();
        }
        String name;
        String age;
        public void init(ServletConfig config)throws ServletException{
                name=config.getInitParameter("name1");
                age=config.getInitParameter("age1");
        }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                ServletContext sc=request.getServletContext();

                out.println("<br><br><b>ServletConfig Parameters</b>");
                out.println("<br><b>Name is:-"+name+"</b>");
                out.println("<br><b>Age is:-"+age+"</b>");

                sc.setAttribute("capital", "new delhi ");
               String email=sc.getInitParameter("email1");
                String country=sc.getInitParameter("country1");
                String cname=sc.getInitParameter("cname1");
                out.println("<br><br><b>ServletContext Parameters</b>");
                out.println("<br><b>Email is:-"+email+"</b>");
                out.print("<br><b>Country name is:- " + country+"</b>");
               out.print("<br><b>Company name is:- " + cname+"</b>");
    }
    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                // TODO Auto-generated method stub
    }

}
```

Servlet 2: CompanyDetails.java

```
public class CompanyDetails extends HttpServlet {
```

```
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
        public CompanyDetails() {
            super();

        }
         /**
          * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
          */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                    response.setContentType("text/html");
                      PrintWriter out = response.getWriter();
                      ServletContext sc = getServletContext();

                      Enumeration e1 = sc.getInitParameterNames();

                      while (e1.hasMoreElements()) {
                        Object obj = e1.nextElement();
                        out.print("<br>" + obj + "  =======   "
                            + sc.getInitParameter(obj.toString()));
                      }
                   out.println("<br>capital is " + sc.getAttribute("capital"));
        }
        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        }
}
```

Deployment Descriptor: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>configncontext</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
```

```
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>default.html</welcome-file>
  <welcome-file>default.htm</welcome-file>
  <welcome-file>default.jsp</welcome-file>
 </welcome-file-list>
 <servlet>
  <description></description>
  <display-name>UserInfo</display-name>
  <servlet-name>UserInfo</servlet-name>
  <servlet-class>example.UserInfo</servlet-class>
  <init-param>
   <param-name>name1</param-name>
   <param-value>Shridhar</param-value>
  </init-param>
  <init-param>
   <param-name>age1</param-name>
   <param-value>25</param-value>
  </init-param>
 </servlet>
 <servlet-mapping>
  <servlet-name>UserInfo</servlet-name>
  <url-pattern>/UserInfo</url-pattern>
 </servlet-mapping>
 <context-param>
  <param-name>email1</param-name>
  <param-value>abc@gmail.com</param-value>
 </context-param>
 <context-param>
  <param-name>country1</param-name>
  <param-value>India</param-value>
 </context-param>
 <context-param>
  <param-name>cname1</param-name>
  <param-value>HCL</param-value>
 </context-param>
 <servlet>
  <description></description>
  <display-name>CompanyDetails</display-name>
  <servlet-name>CompanyDetails</servlet-name>
  <servlet-class>example.CompanyDetails</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>CompanyDetails</servlet-name>
  <url-pattern>/CompanyDetails</url-pattern>
 </servlet-mapping>
</web-app>
```

4. ServletRequest Interface:

ServletRequest interface is used to provide client request information to the servlet. Servlet container creates ServletRequest object from client request and pass it to the servlet service() method for processing. 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.

Some of the important methods of ServletRequest interface are:

Method	Description
Object getAttribute(String name)	This method returns the value of named attribute as Object and null if it's not present. We can use getAttributeNames() method to get the enumeration of attribute names for the request. This interface also provide methods for setting and removing attributes
void setAttribute(String name, Object o)	Stores an attribute in this request
Enumeration getAttributeName()	Return an Enumeration containing the names of the attributes available inthis request
void removeAttribute(String name)	Removes an attribute from this request
String getParameter(String name)	This method returns the request parameter as String. We can use getParameterNames() method to get the enumeration of parameter names for the request.
Enumeration getParameterNames()	Returns an enumeration of all parameter names
String[] getParameterValues(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
String getServerName()	Returns the hostname of the server.
int getServerPort()	Returns the port number of the server on which it's listening. The child interface of ServletRequest is HttpServletRequest that contains some other methods for session management, cookies and authorization of request.
int getContentLength()	Return size of request body
int getContentType()	Return media type of request content
ServletInputStream getInputStream()	Returns a input stream for reading binary data.

String getLocalAddr()	Returns the Internet Protocol(IP) address of the interface on which the request was received.
ServletContext getServletContext()	Return the servlet context of current request.
boolean isSecure()	Returns a boolean indicating whether this request was made using a secure channel, such as HTTPS.

Link for API: http://docs.oracle.com/javaee/6/api/javax/servlet/ServletRequest.html

5. ServletResponse Interface:

ServletResponse interface is used by servlet in sending response to the client. Servlet container creates the ServletResponse object and pass it to servlet service() method and later use the response object to generate the HTML response for client.

Some of the important methods in ServletResponse are:

Methods	Description
PrintWriter getWriter()	Returns a PrintWriter object that can send character text to the client.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response
void setContentLength(int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header
Void setContentType(String type)	Sets the content type of the response being sent to the client before sending the respond.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response.
boolean isCommitted()	Returns a boolean indicating if the response has been committed
void setLocale(Locale loc)	Sets the locale of the response, if the response has not been committed yet.

Link for API:
https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/ServletResponse.html

Example for request and response -

- Below program uses request and response object to take values from the user and display it.
- For this one jsp file is created which pass the values for servlet.
- Servlet take user input with request object and pass response by using response object.

Servlet: RequestResponseExample.java

```
public class RequestResponseExample extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
            public Example() {
               super();
            }
        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        }
        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                PrintWriter out=response.getWriter();
                try{
                        String user=request.getParameter("user");
                        out.println("<h2> Welcome  "+user+"</h2>");
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Jsp file: UserDetails.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Example" method="post">
<table>
<caption>user details</caption>
```

```
<tr>
<td><input type="text" name="user"></td>
<td><input type="submit" value="Submit"></td>
</tr>
</table>
</form>
</body>
</html>
```

6. RequestDispatcher Interface:

RequestDispatcher interface is used to forward the request to another resource that can be HTML, JSP or another servlet in the same context. 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. We can also use this to include the content of another resource to the response. This interface is used for servlet communication within the same context.

There are two methods defined in this interface:

Methods	Description
void forward(ServletRequest request, ServletResponse response)	Forwards the request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
void include(ServletRequest request, ServletResponse response)	Includes the content of a resource (servlet, jsp page, html file) in the response.

We can get RequestDispatcher in a servlet using ServletContext *getRequestDispatcher(String path)* method. The path must begin with a / and is interpreted as relative to the current context root.

Example for request dispatcher:

- Request dispatcher used to open another page from servlet.
- In below program servlet file is created to take response from one jsp page and it checks value for which have provided by the user.
- If that values are correct then it tranfers response to the Welcome.jsp by using request dispatcher.

Servlet: RequestDispatchDemo.java

```
public class RequestDispatchDemo extends HttpServlet {
        private static final long serialVersionUID = 1L;
    /**
     * Default constructor.
     */
```

```
   public RequestDispatchDemo() {
      // TODO Auto-generated constructor stub
   }
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
         }

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

                  response.setContentType("text/html, charset=UTF-8");
                  PrintWriter out=response.getWriter();
                  try  {
                           String name = request.getParameter("uname");
                     String password = request.getParameter("pass");

                     if(name.equals("ctuser01"))
                     {
                        if(password.equals("c0d3r123"))
                        {
                                 RequestDispatcher rd =
request.getRequestDispatcher("Welcome.jsp");
                                 rd.forward(request, response);
                        }
                     }
                     else
                     {
                        out.println("<font color='red'><b>You have entered incorrect
password</b></font>");
                           RequestDispatcher rd =
request.getRequestDispatcher("Registration.jsp");
                           rd.include(request, response);
                     }
                   }
                  catch(Exception e) {
                           e.printStackTrace();S
                  }
                  finally {
                           out.close();
                  }
         }
}
```

Jsp file: Registrations.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Requestdis" method="post">
<table>
<tr><td>Name:</td><td><input type="text" name="uname"/></td>
<tr><td>Password:</td><td><input type="password" name="pass"/></tr>
<tr><td><input type="submit" value="Submit"/></td></tr>
</table>
</form>
</body>
</html>
```

Jsp file: Welcome.jsp:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>Welcome to Squad</h2>
</body>
</html>
```

Difference between forward() and sendRedirect() method:

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

7. GenericServlet class:

GenericServlet is an **abstract class** that implements Servlet, ServletConfig and Serializable interface. GenericServlet provide default implementation of all the Servlet life cycle methods and ServletConfig methods and makes our life easier when we extend this class, we need to override only the methods we want and rest of them we can work with the default implementation. Most of the methods defined in this class are only for easy access to common methods defined in Servlet and ServletConfig interfaces. One of the important method in GenericServlet class is no-argument init() method and we should override this method in our servlet program if we have to initialize some resources before processing any request from servlet.

8. HTTPServlet class:

HTTPServlet is an abstract class that extends GenericServlet and provides base for creating HTTP based web applications. There are methods defined to be overridden by subclasses for different HTTP methods.

There are many methods in HttpServlet class. They are as follows:

Methods	Description
public void service(ServletRequest req,ServletResponse res)	Dispatches the request to the protected service method by converting the request and response object into http type.
protected void doGet(HttpServletRequest req, HttpServletResponse res)	Handles the GET request. It is invoked by the web container.
protected void doPost(HttpServletRequest req, HttpServletResponse res)	Handles the POST request. It is invoked by the web container.
protected void doHead(HttpServletRequest req, HttpServletResponse res)	Handles the HEAD request. It is invoked by the web container.
protected void doOptions(HttpServletRequest req, HttpServletResponse res)	Handles the OPTIONS request. It is invoked by the web container.
protected void doPut(HttpServletRequest req, HttpServletResponse res)	Handles the PUT request. It is invoked by the web container.
protected void doTrace(HttpServletRequest req, HttpServletResponse res)	Handles the TRACE request. It is invoked by the web container.
protected void	Handles the DELETE request. It is invoked by the web

doDelete(HttpServletRequest req, HttpServletResponse res)	container.
protected long getLastModified(HttpServletRequest req)	Returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

9. HttpServletRequest Interface:

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).

Some HttpServletRequest Methods:

Methods	Description
String getContextPath()	Returns the portion of the request URI that indicates the context of the request.
Cookies getCookies()	Returns an array containing all of the Cookie objects the client sent with this request
String getQueryString()	Returns the query string that is contained in the request URL after the path.
HttpSession getSession()	Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.
String getMethod()	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
Part getPart(String name)	Gets the Part with the given name.
String getPathInfo()	Returns any extra path information associated with the URL the client sent when it made this request.
String getServletPath()	Returns the part of this request's URL that calls the servlet.

10. HttpServletResponse Interface:

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).

Some Important Methods of HttpServletResponse:

Methods	Description
void addCookie(Cookie cookie)	adds the specified cookie to the response.
void sendRedirect(String	Sends a temporary redirect response to the client using the

location)	specified redirect location URL and clears the buffer
int getStatus()	gets the current status code of this response
String getHeader(String name)	gets the value of the response header with the given name.
void setHeader(String name, String value)	sets a response header with the given name and value
void setStatus(int sc)	sets the status code for this response
void sendError(int sc, String msg)	sends an error response to the client using the specified status and clears the buffer

11. Filter Interface:

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

Example that have been identified for this design are

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. Since: Servlet 2.3

Some Methods of Filter Interface:

Method	Description
void destroy()	Called by the web container to indicate to a filter that it is being taken out of service.
void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)	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.
void init(FilterConfig filterConfig)	Called by the web container to indicate to a filter that it is being placed into service.

3. SESSION

3.1. Session Management in Servlet:

HTTP is a stateless protocol. All requests and responses are independent. But sometimes we need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.
A session is a conversation between the server and a client. A conversation consists series of continuous request and response.

Session Management is a mechanism used by the Web container to store session information for a particular user. There are four different techniques used by Servlet application for session management. They are as follows:

1. HttpSession
2. Cookies
3. Hidden form field
4. URL Rewriting

Session is used to store everything that we can get from the client from all the requests the client makes. The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

3.2. HttpSession:

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object. Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.

- On client's first request, the Web Container generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
- The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
- The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.

Servlet

1. HttpSession Interface:

Create new Session:

HttpSession session=request.getSession(); // getSession method returns the session. If the session is already exist it return existion session else it creats new session

HttpSession session =request.getSession(true); //returns always new session.

HttpSession session= request.getSession(false); // returns always pre-existing session.

Destroying Session :

session.invalidate(); // destroy the session

Some Important Methods of HttpSession:

Methods	Description
long getCreationTime()	returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT.
String getId()	returns a string containing the unique identifier assigned to the session.
long getLastAccessedTime()	returns the last time the client sent a request associated with the session
int getMaxInactiveInterval()	returns the maximum time interval, in seconds.
void invalidate()	destroy the session
boolean isNew()	returns true if the session is new else false
void setMaxInactiveInterval(int interval)	Specifies the time, in seconds,after servlet container will invalidate the session.

Example of HttpSession:

Servlet: Sessionexample.java

```
/**
 * Servlet implementation class Sessionexample
 */
@WebServlet("/Sessionexample")
public class Sessionexample extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
```

```
    * @see HttpServlet#HttpServlet()
    */
   public Sessionexample() {
      super();
   }

         /**
          * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
          */
         protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
         }

         /**
          * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
          */
         protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                  PrintWriter out=response.getWriter();
                  HttpSession session=request.getSession();
                  String name=request.getParameter("uname");
                  String pass=request.getParameter("pass");
                  if(pass.equals("1234"))
                  {
                           session.setAttribute("uname",name);
                           session.setAttribute("password",pass);
                           out.println("<h2><font color='purple'>Hello"+"  "+name+"</font></h2>");
                  }
                  else
                  {
                           out.println("<font color='red'><b>You have entered incorrect
password</b></font>");
                            RequestDispatcher rd = request.getRequestDispatcher("userdata.jsp");
         rd.include(request, response);
                  }
         }
}
```

Jsp file: Userdata.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Insert title here</title>
</head>
<body>
<form action="Sessionexample" method="post">
<table>
<tr><td>Name:</td><td><input type="text" name="uname"/></td>
<tr><td>Password:</td><td><input type="password" name="pass"/></tr>
<tr><td><input type="submit" value="Submit"/></td></tr>
</table>
</form>
</body>
</html>
```

2. Difference between request.setAttribute and session.setAttribute

The web container handles the request like following. The web container

- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

Thus if any attribute is set in request.setAttribute, its scope is only for that single request until it gets served, after that request attribute get deleted. But if any attribute is set in session its scope remains for multiple requests.

3.3. Cookies for Session Management:

Cookies are small pieces of information that are sent in response from the web server to the client. Cookies are the simplest technique used for storing client state. Cookies are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan. Using Cookies for storing client state has one shortcoming though, if the client has turned of COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

1. Cookies API:

Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response** object using the addCookie() method. This method sends cookie information over the HTTP response stream. getCookies() method is used to access the cookies that are added to response object.

2. Types of Cookies:

Servlet

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie:

It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie:

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Some commonly used methods of the Cookie class:

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Example of Cookies:

Jsp file: Sample.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="Cookies" method="post">
Enter Name: <input type="text" name="uname" id="uname"><br>
<input type="submit" value="Submit" >
</form>
</body>
</html>
```

Servlet 1: FirstServlet.java

```
/**
 * Servlet implementation class Cookies
 */
@WebServlet("/Cookies")
public class Cookies extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Cookies() {
        super();
    }
	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

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

		String n =request.getParameter("uname");
		out.println("welcome"+" "+n);

		Cookie ck=new Cookie("username",n);
		response.addCookie(ck);

		response.sendRedirect("Cookies1");
	       out.close();

	}
}
```

Servlet 2: Cookies1.java

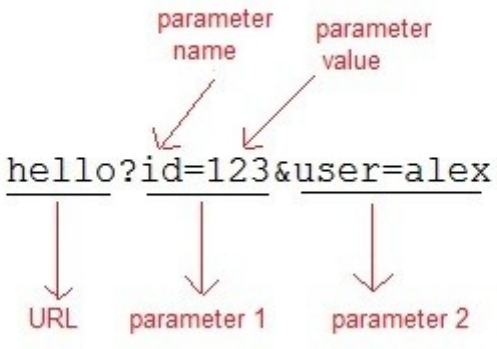
```
/**
 * Servlet implementation class Cookies1
```

```
 */
@WebServlet("/Cookies1")
public class Cookies1 extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Cookies1() {
        super();

    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

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

		Cookie ck1[]=request.getCookies();
		out.print("Hello"+"  "+ck1[1].getValue());
		out.print("<br>Cookie name"+"  "+ck1[1].getName());
		 out.close();
		 }

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

	}

}
```

3.4. URL rewriting in Session Management:

If the client has disabled cookies in the browser then session management using cookie wont work. In that case URL Rewriting can be used as a backup. URL rewriting will always work. In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal(=) sign.

For example,



When the User clicks on the URL having parameters, the request goes to the Web Container with extra bit of information at the end of URL. The Web Container will fetch the extra part of the requested URL and use it for session management.

The getParameter() method is used to get the parameter value at the server side.

Example:

Jsp file: Urlwriting.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<br>
Enter first number:
<input type="text" name="fnum" value="12"><br>
Enter Second number:
<input type="text" name="snum" value="15"><br>

<a href="Operation?firstnum=12&secondnum=15">Click here to see addition</a>
</body>
</html>
```

Servlet: Operation.java

```
/**
 * Servlet implementation class Operation
 */
@WebServlet("/Operation")
public class Operation extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Operation() {
        super();
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		int n1=Integer.parseInt(request.getParameter("firstnum"));
		int n2=Integer.parseInt(request.getParameter("secondnum"));
		int add=n1+n2;
		PrintWriter out=response.getWriter();
		out.println("Addition is:"+add);
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
	}

}
```

3.5. Hidden Form Field:

Hidden form field is used to store the state of the user. It uses invisible textfield to store the value which we want to refer in the servlet. Whithout depending on the browser we can pass values to the servlet.

Syntax :

<input type=”hidden” name=”action” value=”delete”>

1. Real application of hidden form field:

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

2. Advantage of Hidden Form Field:

1. It will always work whether cookie is disabled or not.

3. Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

Example:

Servlet: Hiddenfield.java

```
/**
 * Servlet implementation class Hiddenfield
 */
@WebServlet("/Hiddenfield")
public class Hiddenfield extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Hiddenfield() {
        super();
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		String name=request.getParameter("uname");
		String id=request.getParameter("userid");
		out.println("<br><b>Your name:"+name);
```

```
                out.println("<br><b>Your id:"+id);
        }
}
```

Jsp file: HiddenFieldDemo.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Hidden Field</title>
</head>
<body>
<form action="Hiddenfield" method="post">
<input type="hidden" name="userid" value="ctuser01">
User Name: <input type="text" name="uname">
<input type="submit" value="Submit">
</form>
</body>
</html>
```

ASSIGNMENT

1. Write a program for servlet to display your name.
2. Write a program to take single parameter from web page.
3. Write a program to retrive configuration information about servlet.
4. Write a servlet that shows current server time and updated in every 10 seconds.
5. Write a servlet that snoops all the information about the current session. Make a “registration” form that collects a first name, last name, and email address. Send the data to a servlet that displays it. Next, modify the servlet to use a default value in form(or give an error message) if the user omits any of the three required parameters.
6. Use session tracking to make a servlet that says ” Welcome Mumbai” to first-time visitors (within a browsing session) and ” Welcome Back ” to return visitors. Do this job using Session Tracking and Cookies. Also compare these 2 approaches with respect to simplicity.
7. Write a program to show inter servlet communication between two servlets.(Servlet chaining).
8. Write a program, using servlet and JDBC which takes students roll number and provides student information, which includes the name of the student, the address, email-id, program of study, and year of admission. You have to use a database to store student’s information.
9. Write program for user log in using login and password. Display a message if login and password are not correctly given.

10. Write a program using servlet and JDBC for developing an online application for the shopping of computer science books. (Hint: use concept of session tracking) You have to create a database for book title, author(s) of book, publisher, year of publication, price. Make necessary assumptions for book shopping.
11. This servlet counts and displays the number of times it has been accessed, and saves the count to a file in its destroy() method to make the count persistent.
12. Write a servlet that checks the client machine and only allows access if the client appears to be coming from somewhere other than the Terrorist countries. Assume some 5 countries as terrorist countries.
13. Write a servlet application that takes employee data from form and save into database, after storing the data display success message – "Employee data saved successfully". The success message will appear on screen for 3 seconds and automatically the page will redirect to employee list to show existing employees (stored in database).
14. Write a servlet that performs a random redirect, sending a client to a random site selected from its existing site list. The site list containing advertising images, it can be used to select the next adv banner.
15. Write a servlet that redirects requests form one host to another host, giving an explanation to the client before the redirection. As we generally see during payment and request will be redirected to bank website.
16. Write a servlet that demonstrates session tracking using hidden form fields by displaying the shopping cart for a booksite.
17. Write a servlet that uses session tracking to count the number of times a client has accessed it.
18. Write a servlet that demonstrates how to programmatically alter the current session timeout. On first execution, the current timeout displays the application-wide setting. On second execution, the current timeout displays two hours—because that's the timeout set during the first execution.(Hint: session max inactive interval setting in web.xml file or programmatically) .
19. Write a servlet that is protected by BASIC authentication as shown in web.xml and tomcat-users.xml. To see the salary information you'll need to login as a "manager" using names and passwords in tomcat-users.xml.
20. Write an news application that displays different categories of news, also provide a form to personalize the news for some given time period (1,2,3 months) (Hint: use cookies) .
21. Write a servlet that lets a user vote for his favorite food from a combo box or radio Buttons (the user must be able to make multiple food selections per request). Store the favorite foods and the number of votes for each food. Please note that a user from one IP address can vote only once. Display all foods and their number of votes in alphabetical order back to the user. Use an appropriate Collection class or Map class to store the data.(or database can also be used to store data).
22. Write a servlet that displays the values of the firstName, lastName, and emailAddress request parameters. But, remember what users told you in the past, and use the old values if the current values are missing. So, if a parameter is missing and the client is a first-time

visitor, have the servlet list “Unknown” for the missing values. If a parameter is missing and the client is a repeat visitor, have the servlet use previously entered values for the missing values. This should definitely be easier than a version that uses cookies explicitly.

23. Make a servlet that prints a list of the URLs of the pages that the current user has used to link to it (within the current browsing session). That is, if one user has followed hypertext links from three different pages to the servlet, the servlet should show the user the URLs of those three pages. Test out your servlet by making a couple of different static Web pages that link to it. If you feel inspired, modify the basic approach from the notes so that you do not store repeated entries; if the same user follows a link from page1 to the servlet twice, the servlet should list the URL of page1 only once in the list. ArrayList/Map data structure can be used to store page names and urls.