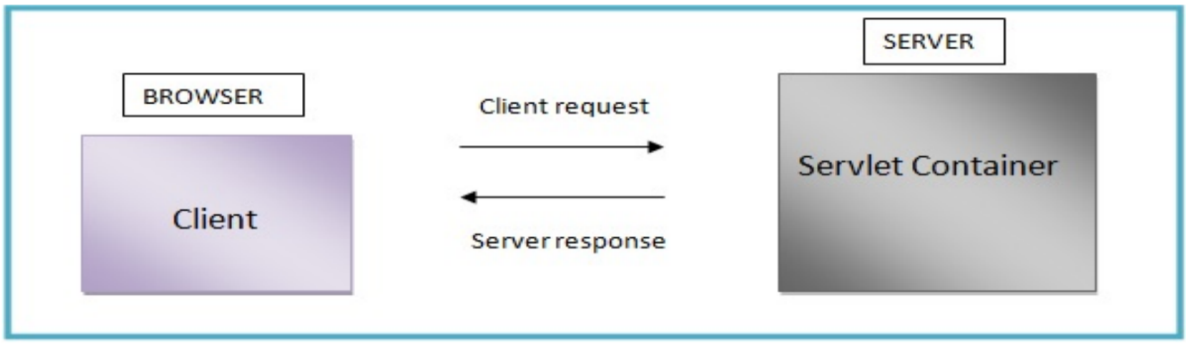
UNIT 2 – PART – 1 – SERVLETS

- Java Servlets are server side Java programs that require either a Web Server or an Application Server for execution.
- Examples for Web Servers include Apache's Tomcat Server and Macromedia's JRun.
- Examples for other Server programs include Java Server Pages (JSPs) and Enterprise Java Beans (EJBs).



- Servlets are server side components that provide a powerful mechanism for developing server side programs.
- Servlets are server as well as platform-independent. Using servlets web developers can create fast and efficient server side application which can run on any servlet enabled web server.
- Servlets run inside the Java Virtual Machine. Since the Servlet runs at server side so it does not check the browser for compatibility.
- It extends from a Java class or rather interface and requires you to implement certain methods so that web container or servlet container (Tomcat, etc.) is able to send execution to your servlet.
- JSP (Java Server Pages), JSF (Java Server Faces), Struts, Spring, Hibernate and others are extensions of the servlet technology.
- Servlets uses the classes in the java packages **javax.servlet and javax.servlet.http**.

Servlet Architecture

Before introducing servlets, CGI (common gateway interface) was used. Servlets facilitate client request and response tasks dynamically. They execute various functions, such as

- Control the flow of the application.
- Generate dynamic web content.
- Server-side load balancing.
- Implement business logic.

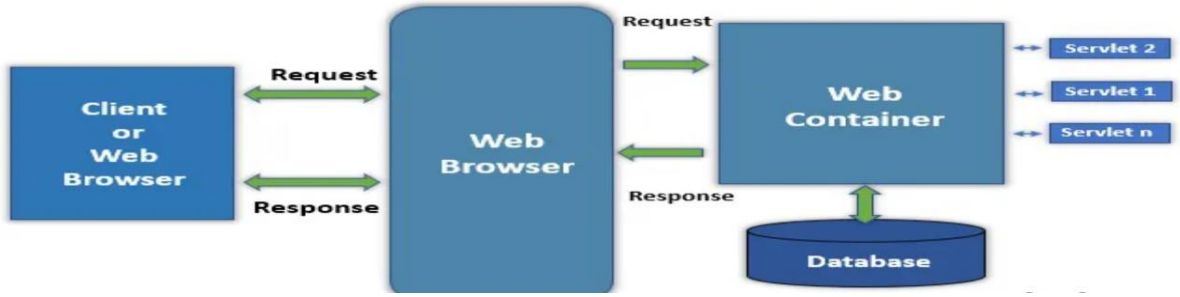
There are two types of Servlets-

1. Generic Servlets
2. HTTPServlets.

Servlets can be created in three ways

(i)Implementing Servlet Interface,
(II)Extending Generic Servlet.
(III)Extending HTTPServlet.

Three life cycle methods available with servlets are **init(), service(), and destroy()**. Every servlet should override these methods.
Below is the diagram to show how components work on servlet architecture.



1. Client

- In this architecture, the web browser acts as a Client. Client or user connected with a web browser.
- The client is responsible for sending requests or HttpRequest to the web server and processing the Web server's responses.

2. Web Server

- The web server controls how web users access hosted files and is responsible for processing user requests and responses.
- Here server is software it understands URLs and HTTP protocol. Whenever a browser needs to host a file on the web server, it processes a client request using an HTTP request; if it finds the requested file, it sends it back to the browser through HTTP Response.

3. Web Container

- A web container is a web server component that interacts with Java servlets. A web container manages the servlets' lifecycle.
- Web container handles the server-side requests of servlets, JSP, and other files.

Servlet Request Flow

The steps to processing a servlet request; consider the above diagram.

- The client sends a request.
- The web server accepts the request and forwards it to the web container.
- Web container **searches web.xml file** for request URL pattern and gets the address of the servlet.
- You should **create and set up the servlet using the init()** method if it has not been created yet.
- The **container calls public service()** by passing ServletRequest and ServletResponse objects.
- Public service() method typecast ServletRequest and ServletResponse objects to HttpServletRequest and HttpServletResponse objects, respectively.
- The public service() method calls for protected service().
- The protected service() method checks the client request & corresponding do___() method is called.
- The request is handled by sending the result generated by do___() to the client.

Servlet API

- It communicates with the clients through a request and response format or system in a servlet container.
- An application programming interface (API) for servlets is a group of predefined packages, classes, and interfaces that have been written, along with their servlet methods and constructors.
- It is a mechanism for interaction between two web application components and servlet functions.
- Servlet API comes in two packages, which are:
    - javax.servlet
    - javax.servlet.http

  The servlet pages or web container uses many interfaces and classes from the "javax.servlet" package. The "javax.servlet.http" package has classes and interfaces that manage http requests.

1. javax.servlet.http package

- **HttpServletRequest:** To deliver servlets details about HTTP client requests. It uses the servlet interface, which is ServletRequest.
- **HttpServletResponse:** To send a response to a service user in a way that works with HTTP. It uses the servlet interface, which is ServletResponse.
- **HttpSession:** It gives a way to keep track of information about a user across an application or website's webpage and identify that client.
- **HttpSessionActivationListener:** The container uses to tell all the objects connected to a session, and the session will be turned off and then turned on.
- **HttpSessionAttributeListener:** This listener interface can be used to find out when changes are made to the attribute lists of sessions in this web application.
- **HttpSessionBindingListener:** It tells a HttpSessionBindingEvent object to send information or a message to an object when that object is bound to the session or unbound from a session.
- **HttpSessionListener:** When something changes in a web page or application's list of active sessions, the interface gets notified.

2. javax.servlet package

The following several Servlet API interfaces use in the "javax.servlet" package:

- **Filter:** To filter the request to a resource, the response from it, or both functions.
- **FilterChain:** To show the developer how a filtered request for a resource went through the invocation chain. This is done by the servlet container.
- **FilterConfig:** To send information to a filter when a servlet container starts up.
- **RequestDispatcher:** It makes an object that sends the request and response to any other resource. This means it takes requests from the client and sends them to a servlet, HTML file, or JSP file on the server.
- **Servlet:** This primary interface tells all servlets what methods they must implement. Write a generic servlet that extends javax.servlet to use this interface. GenericServlet or a servlet for HTTP that extends javax.servlet. http.HttpServlet.
- **ServletConfig:** It describes an object created by a servlet page when a servlet is instantiated and is used to send information to the servlet when it is initialized.

- **ServletContext:** It describes a group of methods that a servlet can use to talk to its servlet container. The ServletContext object is where the information in the web.xml file about the web application is kept.
- **ServletContextAttributeListener:** The classes use this interface when the attribute list of a web application's servlet context changes.
- **ServletContextListener:** The classes that use this interface are told when the servlet context of the web application they are a part of changes.
- **ServletRequest:** It describes an object a servlet container makes to pass information about a client's request to a servlet.
- **ServletRequestAttributeListener:** Let the listener know when a request attribute changes while the request is in the scope of the web application where the listener is registered.
- **ServletRequestListener:** Let a web component know when requests come in and leave its scope.
- **ServletResponse:** It describes an object a servlet container makes to help a servlet send a response to a service user.

Servlet Life Cycle:-

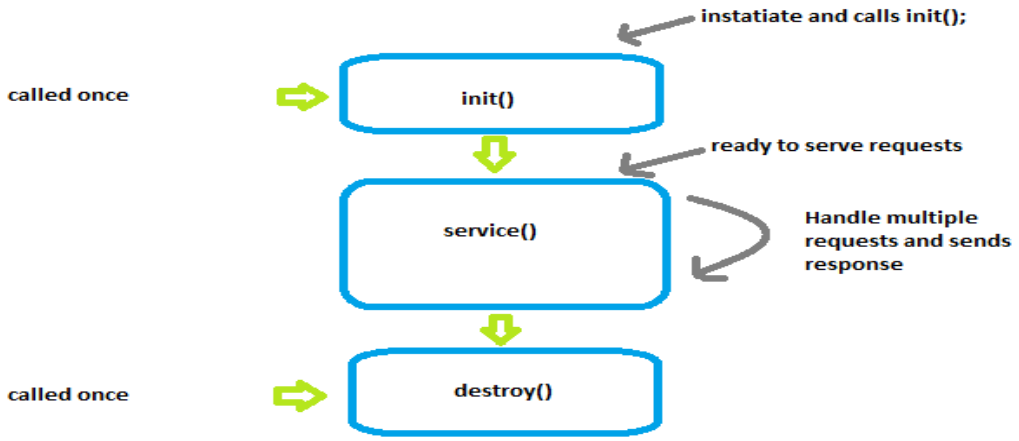
Servlets are the Java programs which run inside a Servlet Container. A Servlet Container is much like a Web Server which handles user requests and generates responses.

It can also contain Java Servlets and JSP pages to generate dynamic response. Servlet Container is responsible for loading and maintaining the lifecycle of the Java Servlet.

Servlet Container can be used standalone or more often used in conjunction with a Web server. Example of a Servlet Container is Tomcat and that of Web Server is Apache.

There are three methods used to maintain the lifecycle of servlet.

1) **init()**
2) **service()**
3) **destroy()**



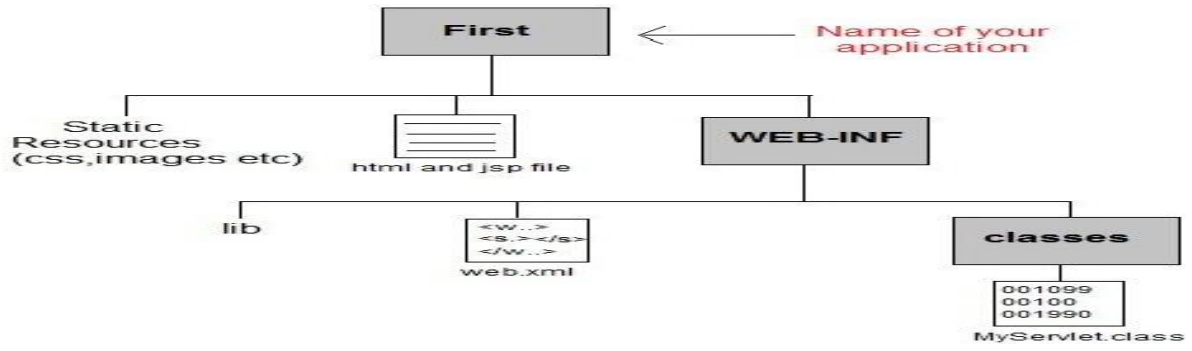
- ✓ **init()**
  - When the servlet is first created, its init() is invoked, so init is where you put one time setup code.
  - It is called when the servlet Is first created, and not called again for each user request, so it is used for one time initialization, just as with the init() of applet.
  - This method is called once when the servlet is loaded into the servlet engine, before the servlet is asked to process its first request.
  - The init method has a ServletConfig parameter. The servlet can read its initialization arguments through the ServletConfig object. It may be usually defined in a configuration file.
- ✓ **service()**
  - Each time the server receives a request for a servlet,the server needs a new thread and calls service().
  - The service () checks the HTTP request type(GET,POST,PUT,DELETE etc) and calls doGet,doPut,doDelete etc.as appropriate.
- ✓ **destroy()**
  - The service may decide to remove a previously loaded servlet instance, perhaps because it is explicitly asked to do so by the server administrator or perhaps because the servlet is idle for a long time.

Developing and Deploying Servlets

- To create a Servlet application, you need to follow the below mentioned steps. These steps are common for all the Web server.
- In example we are using Apache Tomcat server.
- **Apache Tomcat is an open source web server for testing servlets and JSP technology.**
- Download latest version of **Tomcat Server** (LINK FOR TOMCAT SERVER)and install it on your machine.
- After installing Tomcat Server on your machine follow the below mentioned steps :
  - ✓ Create directory structure for your application.
  - ✓ Create a Servlet
  - ✓ Compile the Servlet
  - ✓ Create Deployment Descriptor for your application
  - ✓ Start the server and deploy the application

1. Creating the Directory Structure

- ✓ Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



- Create the above directory structure inside Apache-Tomcat\webapps directory.
- All **HTML, static files(images, css etc) are kept directly under Web application folder. While all the Servlet classes are kept inside classes' folder.**
- **The web.xml (deployment descriptor) file is kept under WEB-INF folder.**

2. Creating a Servlet

There are three different ways to create a servlet.

- ✓ By implementing Servlet interface
- ✓ By extending GenericServlet class
- ✓ By extending HttpServlet class

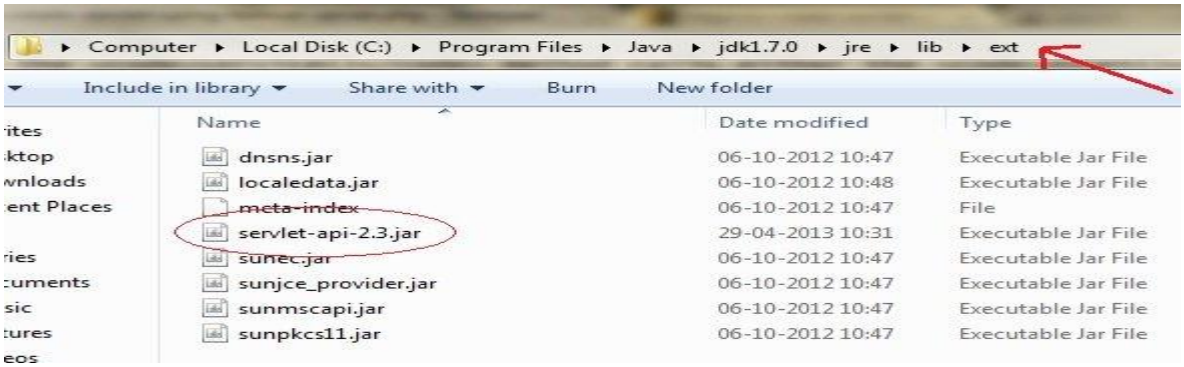
- Our servlet class will override only doGet() or doPost() method also client's methods.
- When a request comes in for the servlet, the Web Container calls the servlet's service() method and depending on the type of request the service() method calls either the doGet() or doPost() method.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public MyServlet extends HttpServlet
{
        public void doGet(HttpServletRequest request,HttpServletResposne response)
                throws ServletException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<html><body>");
            out.println("<h1>Hello Readers</h1>");
            out.println("</body></html>");    } }
```

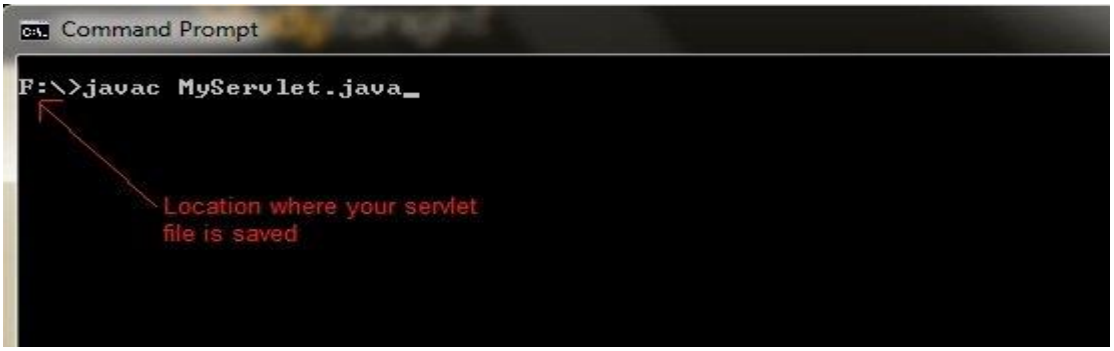
- Write above code in a notepad file and save it as MyServlet.java anywhere on your PC. Compile it(explained in next step) from there and paste the class file into WEB-INF/classes/ directory that you have to create inside Tomcat/webapps directory.

3. Compiling a Servlet

- To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class
- Steps to compile a Servlet:



- Set the Class Path
- Download servlet-api.jar file.
- Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.
- Compile the Servlet class.



NOTE: After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

4. Create Deployment Descriptor

- Deployment Descriptor (DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:
  - ✓ Mapping URL to Servlet class.
  - ✓ Initializing parameters.
  - ✓ Defining Error page.
  - ✓ Security roles.
  - ✓ Declaring tag libraries.
- Now we will see how to create a simple web.xml file for our web application.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
     <servlet>
         <servlet-name>hello</servlet-name>
         <servlet-class>MyServlet</servlet-class>
     </servlet>
     <servlet-mapping>
         <servlet-name>hello</servlet-name>
         <url-pattern>/hello</url-pattern>
     </servlet-mapping>

</web-app>
```

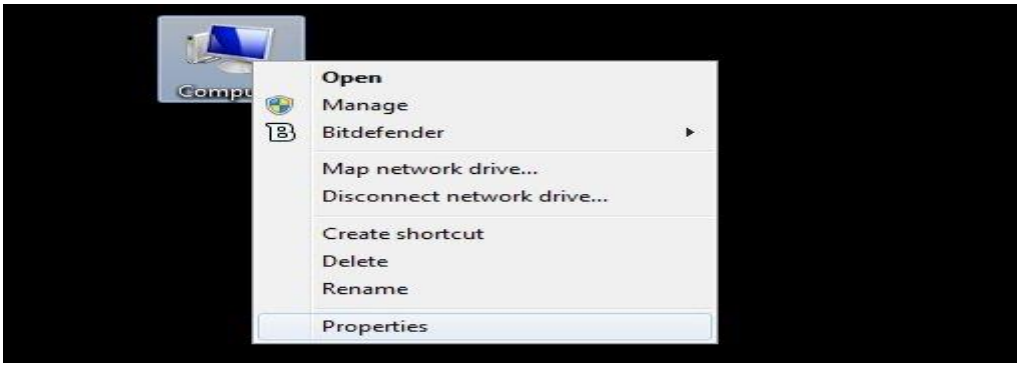
5. Start the Server

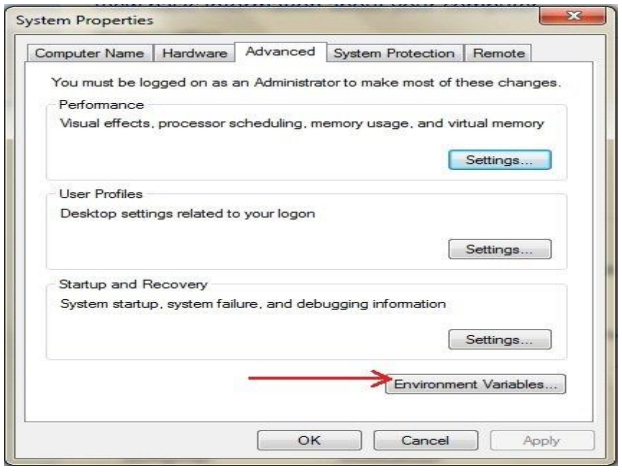
- Double click on the startup.bat file to start your Apache Tomcat Server.
- Or, execute the following command on your windows machine using RUN prompt.

```
C:\apache-tomcat-7.0.14\bin\startup.bat
```

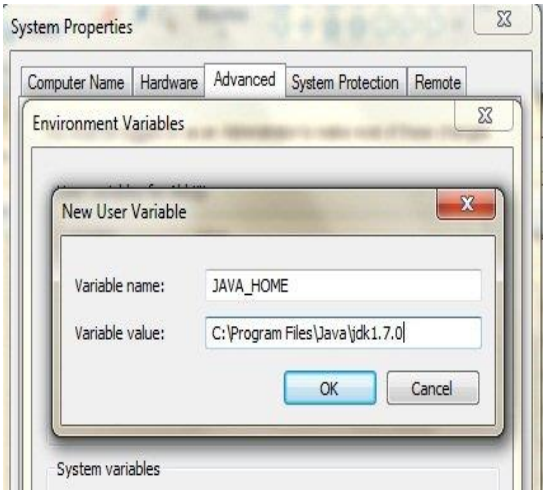
6. Starting Tomcat Server for the first time

- If you are starting Tomcat Server for the first time you need to set JAVA_HOME in the Environment variable. The following steps will show you how to set it.
  - Right Click on My Computer, go to Properites.
  - Go to Advanced Tab and Click on Enviroment Variables... button.





- Click on New button, and enter JAVA_HOME inside Variable name text field and path of JDK inside Variable value text field. Click OK to save.



7. Run Servlet Application

Open Browser and type http:localhost:8080/First/hello

Servlet Request and Response

Servlet handles various client requests and provides their responses. It provides two interfaces i.e ServletRequest and ServletResponse for this purpose.

ServletRequest interface

- ✓ ServletRequest is an interface whose object is used to provide the information of each request to servlet.
- ✓ This information may be any name, type, value or other attribute. This interface is present in javax.servlet package.
- ✓ Servlet container is responsible to create ServletRequest object which is given with service() method argument.

Methods of ServletRequest

Method	Description
String getParameter(String name)	This method returns the value given in request as a String.
Object getAttribute(String name)	This method provides the attribute of request as an Object.
String getServerName()	This method provides the server name to which request is sent.
int getServerPort()	This method returns the port number to which request is sent.
boolean isSecure()	This method indicates whether the server is secure or not.

ServletResponse interface

- ✓ The object of ServletResponse interface is used to send the responses to the clients.
- ✓ The information send in responses can be a binary or character data.
- ✓ ServletResponse interface is present in javax.servlet package and passes as an arguement of service() method.

Methods of ServletResponse

Method	Description
PrintWriter getWriter()	This method is used to send character data in response.
int getBufferSize()	This method returns the capacity of buffer in sent response.

void reset()	This method is used to remove the data present in buffer.
void setContentType(String type)	This method is used to set the type of content.

HttpServletRequest Class

Methods	**Description**
Enumeration getHeaders()	Returns all the values of the specified request headers as an Enumeration.
Cookies[] getCookies()	Used to obtain the array of cookie that are present in the request.
String getQueryString()	Returns the query string that is contained in the request URL.
HttpSession getSession()	Returns the current session associated with the request.

HttpServletResponse Class

Methods	**Description**
Void addCookie(Cookie cookie)	Adds cookie to specified HTTP response.
String encodeURL(String url)	It encodes the URL by including session id in it, if encoding is not needed returns URL unchanged.
void sendRedirect(String url) throws IOException	Redirects the client to url.
void setHeader(String name, String value)	Used to set response header with given name and value.
void setStatus(int sc)	Sets the status code for HTTP response SC_MOVED_TEMPORALILY,SC_OK

Servlet Request and Response example with HTML

index.html

```
<form action="serv" method="get">
<h3>Enter your Name:</h3>
<input type="text" name="username">
<input type="submit" value="submit">
</form>
```

HttpServletReq.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HttpServletReq extends HttpServlet
{
 public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException
 {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        String s1=req.getParameter("username");
```

```
pw.println("<h1>Welcome:"+s1+"</h1>");
String s2=req.getServerName();
pw.println("<h1>Server name:"+s2+"</h1>");
int i=req.getServerPort();
pw.println("<h1>Server Port:"+i+"</h1>");
boolean b=req.isSecure();
pw.println("<h1>Is the server secure:"+b+"</h1>");
}
}
```

Reading Initialization Parameters in Servlet

Both **ServletContext** and **ServletConfig** are basically the configuration objects which are used by the servlet container to initialize the various parameter of a web application.

Difference between ServletConfig vs. ServletContext

ServletConfig	ServletContext
ServletConfig object is one per servlet class.	ServletContext object is global to the entire web application.
Object of ServletConfig will be created during the initialization process of the servlet.	Object of ServletContext will be created at the time of web application deployment
We have to give the request explicitly in order to create the ServletConfig object for the first time.	ServletContext object can be available even before giving the first request.
Scope: As long as a servlet is executing, the ServletConfig object will be available, it will be destroyed once the servlet execution is completed.	Scope: As long as a web application is executing, the ServletContext object will be available, and it will be destroyed once the application is removed from the server.
getServletConfig() method is used to obtain Servletconfig object.	getServletContext() method is used to obtain ServletContext object.

Session Tracking in Servlet

- ✓ Session tracking is a mechanism that servlets use to maintain state about a series of requests from the same user (that is, requests originating from the same browser) across some period of time. It is also known as session management in servlet.
- ✓ Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of a user to recognize to particular user.
- ✓ Session Tracking Techniques
  - There are four techniques used in Session tracking:

    1. Hidden Form Field
    2. URL Rewriting
    3. Cookies
    4. HttpSession

1. Hidden Form Field

- ✓ In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of a user.

- ✓ In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
    - <input type="hidden" name="uname" value="testing">

Advantage of Hidden Form Field

It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

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

2. URL Rewriting

- ✓ In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:
    - url?name1=value1&name2=value2&?
- ✓ 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.

Advantage of URL Rewriting

- ✓ It will always work whether cookie is disabled or not (browser independent).
- ✓ Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- ✓ It will work only with links.
- ✓ It can send only textual information.

3. Cookies

- ✓ A cookie is a piece of data from a website that is stored within a web browser(Client side) that the website can retrieve at a later time.
- ✓ Cookies are used to tell the server that users have returned to a particular website.

Commonly used Methods in Cookie

1) **public String getName( )** Returns the name of the cookies.
2) **public String getValue( )** Returns the value of the cookie.
3) **public int getMaxAge( )** Returns the maximum age limit to the cookie, specified in seconds.
4) **public void setMaxAge(int expiry)** Sets the maximum age of the cookies in seconds.
5) **public void setValue(String newValue)** Allocates a new value to a cookie after the cookie is created.

Create a Cookie Object

The constructor of Cookie class creates the cookie object with corresponding cookie name and value.

```
//creating cookie object
```

```
Cookie cookie = new Cookie("username","testing");
//adding cookie to the response
Response.addCookie(cookie);
```

Get Cookie from client request

```
Cookie ck[ ]=request.getCookies();
```

Get Cookie from client request

```
Cookie ck[ ]=request.getCookies();
```

cookiedemo.html

```
<html>
        <head>
                <title>Cookie Demo</title>
        </head>
        <body>
        <form action="servlet1" method="post">
        Enter Your Name:<input type="text" name="userName"/><br/><br/>
        <input type="submit" value="SUBMIT"/>
                </form>
        </body>
</html>
```

SetCookieServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetCookieServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response){
      try{
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         String name=request.getParameter("userName");
         //creating cookie object
         Cookie ck=new Cookie("uname",name);
         //adding cookie in the response
         response.addCookie(ck);
         out.print("Cookie Saved");
         //creating submit button
         out.print("<br><form action='servlet2' method='post'>");
         out.print("<input type='submit' value='View the cookie value'>");
         out.print("</form>")
         out.close();
      }
      catch(Exception e){
```

```
        System.out.println(e);      }   }
}
```

GetCookieServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetCookieServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response){
      try{
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         Cookie ck[]=request.getCookies();
         out.println("<h3>Reading the cookies</h3>");
         out.println("<br/>");
         out.println("Name of the cookie : " + ck[0].getName() + "<br/>");
         out.println("Value in cookie : " + ck[0].getValue());
         out.close();
      }catch(Exception e){
         System.out.println(e);
      }}}
```

web.xml

```
<web-app>
        <servlet>
                <servlet-name>set</servlet-name>
                <servlet-class>SetCookieServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>set</servlet-name>
                <url-pattern>/servlet1</url-pattern>
        </servlet-mapping>
        <servlet>
                <servlet-name>get</servlet-name>
                <servlet-class>GetCookieServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>get</servlet-name>
                <url-pattern>/servlet2</url-pattern>
        </servlet-mapping>
</web-app>
```

4. HTTPSession in Servlet

- ✓ A session contains information specific to a particular user across the whole application.
- ✓ When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session.
- ✓ The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file(web.xml).
- ✓ The default timeout value is 30 minutes, this is used if you don't specify the value in tag.

Commonly used Methods of Servlet HttpSession

1) **setAttribute(String name, Object value)**: Binds an object to this session, using the name specified.
2) **getAttribute(String name)**: Returns the object bound with the specified name in this session, or null if no object is bound under the name.
3) **getAttributeNames()**: Returns an Enumeration of String objects containing the names of all the objects bound to this session.
4) **getId()**: Returns a string containing the unique identifier assigned to this session.
5) **removeAttribute(String name)**: Removes the object bound with the specified name from this session.

sessiondemo.html

```
<html> <head> <title>session Demo</title> </head>
        <body>
                <form action="servlet1" method="post">
        Enter Your Name:<input type="text" name="userName"/><br/><br/>
        <input type="submit" value="SUBMIT"/>
        </form> </body></html>
```

SetSessionServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetSessionServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name=request.getParameter("userName");
      // Create a session object.
        HttpSession session = request.getSession(true);
        session.setAttribute("uname",name);
       //creating submit button
        out.print("<br><form action='servlet2' method='post'>");
        out.print("<input type='submit' value='View the session'>");
```

```
            out.print("</form>");
            out.close();
        }
        catch(Exception e){
           System.out.println(e);       }}}
```

GetSessionServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetSessionServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response){
      try{
            response.setContentType("text/html")
            PrintWriter out = response.getWriter();
            HttpSession session=request.getSession(false);
            String name=(String)session.getAttribute("uname");
            out.print("Hello!.. Welcome "+name);
            out.close();
      }catch(Exception e){
         System.out.println(e);}}}
```

web.xml

```
<web-app>
            <servlet>
                        <servlet-name>set</servlet-name>
                        <servlet-class>SetSessionServlet</servlet-class>
            </servlet>
            <servlet-mapping>
                        <servlet-name>set</servlet-name>
                        <url-pattern>/servlet1</url-pattern>
            </servlet-mapping>
            <servlet>
            <servlet-name>get</servlet-name>
                        <servlet-class>GetSessionServlet</servlet-class>
            </servlet>
            <servlet-mapping>
                        <servlet-name>get</servlet-name>
                        <url-pattern>/servlet2</url-pattern>
            </servlet-mapping>
</web-app>
```