Unit 6

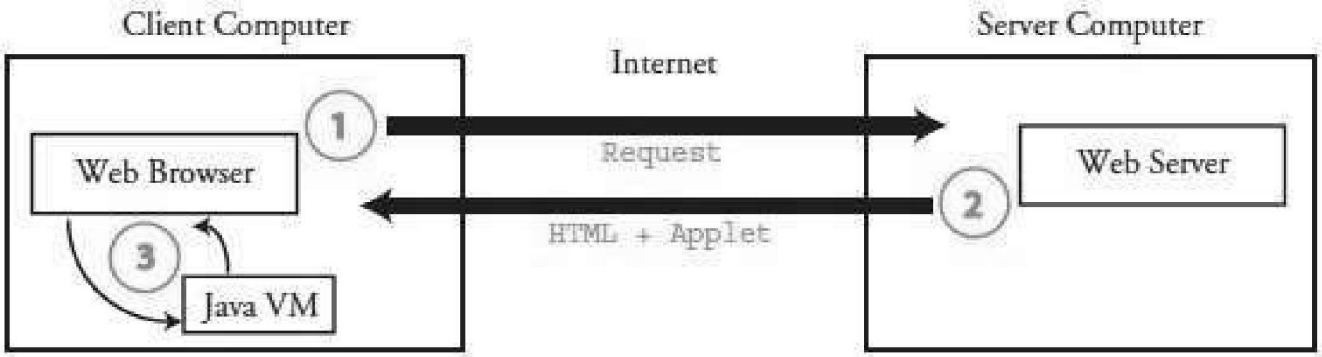
Servlets and JSP 6hrs

6.1 Overview of Web Application

When we instruct our web browser to view a page from a web server on the Internet, our web browser requests the page from the Web server, the Web server processes the request (which may involve reading the requested page from a file on the hard drive), and then the Web server sends the requested page to our Web browser. Our Web browser *formats*, or *renders*, the *received data to fit on our computer screen*. This interaction is a specific case of the ***client/server*** model. Our Web browser is the client program, our computer is the *client computer*, the remote website is the *server computer*, and the Web server software running on the remote website is the *server program*.

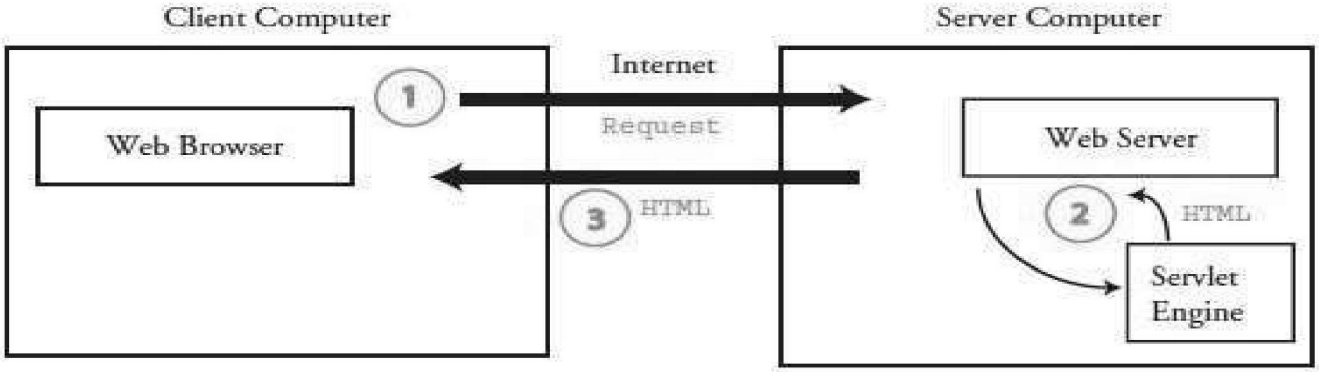
In the context of a Web application, the *client/server* model is important because Java code can run in two places: on the ***client*** or on ***the server***. There are trade-offs to both approaches. *Server-based programs have easy access to information that resides on the server*, such as customer orders or inventory data. Because all of the computation is done on the server and results are transmitted to the client as HTML, a client does not need a powerful computer to run a server-based program. On the other hand, a *client-based* program may require a more powerful client computer, because all computation is performed locally. However, richer interaction is possible, because the client program has access to local resources, such as the *graphics display* (e.g., perhaps using Swing) or the operating system. Many systems today are constructed using code that runs on both the client and the server to reap the benefit of both approaches.

Web applications built with Java include ***Java applets, Java servlets***, and ***Java Server Pages (JSP)***. Java applets run on the *client computer*. *JavaScript*, which is a different language than Java despite its similar name, also runs on the client computer as part of the Web browser. ***Java servlets*** and ***Java Server Pages*** run on the ***server***. *Java Server pages* which are a dynamic version of *Java servlets*. *Servlets must be compiled before they can run, just like a normal Java program*. In contrast, *JSP code is embedded with the corresponding HTML and is compiled "on the fly"* into a servlet when the page is requested. This flexibility can make it easier to develop Web applications using JSP than with Java servlets. The following fig1, fig2 and fig2 shows that the running a Java applet, Java Servlet and JSP program.



1. The client's Web browser sends a request to the server for a Web page with a Java applet.
2. The server sends the HTML for the Web page and applet class files to the client.
3. The client runs the applet using the Java Virtual Machine and displays its output in the Web browser.

Fig1: Running a Java Applet



1. The client's Web browser sends a request to the server for a Web page that runs a Java servlet.
2. The Web server instructs the servlet engine to execute the requested servlet, which consists of running precompiled Java code. The servlet outputs HTML that is returned to the Web server.
3. The Web server sends the servlet's HTML to the client's Web browser to be displayed.

Fig2: Running a Java Servlet

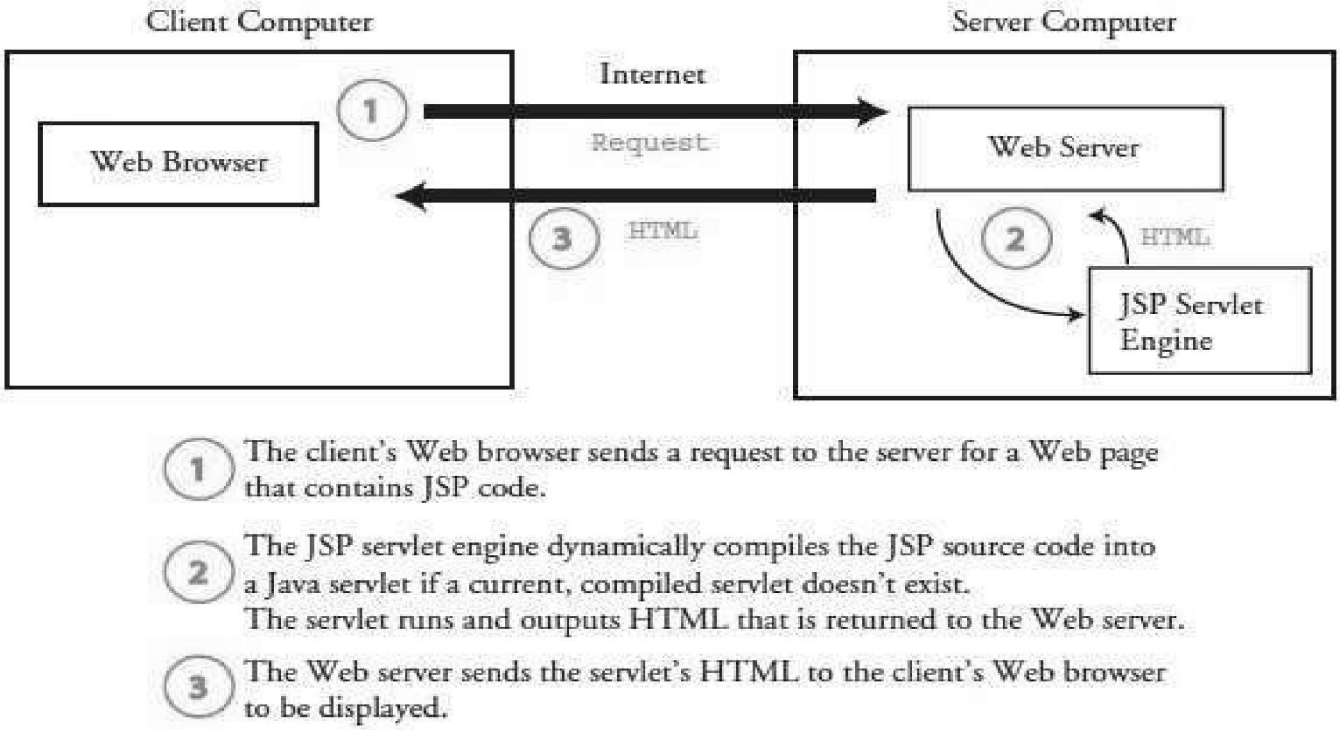


Fig3: Running a Java Server Page (JSP) Program

Advantages of Servlet

Servlets offer several advantages in comparison with traditional system.

1. ***First**, performance is significantly better*. Servlets execute within the *address space* of a web server. It is *not necessary* to *create a separate process to handle each client request*.
2. **Second**, *servlets are platform-independent because they are written in Java*.
3. **Third**, *the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine*.
4. **Finally**, *the full functionality of the Java class libraries is available to a servlet*. It can communicate with *applets, databases, or other software* via the sockets and RMI mechanisms.

6.2 HTTP Methods and Responses

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server.

Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

HTTP Methods

- GET

- POST
- PUT
- HEAD
- DELETE
- PATCH
- OPTIONS
- CONNECT
- TRACE

The two most common HTTP methods are: GET and POST.

The GET Method

GET is used to request data from a specified resource.

Note that the query string (name/value pairs) is sent in the URL of a GET request:

https://www.nast.edu.np/be-computer?name1=value1&name2=value2

Some notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

The POST Method

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the request body of the HTTP request:

POST /be-computer/signup.php HTTP/1.1
Host: nast.edu.np

name1=value1&name2=value2

Some notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Compare GET vs. POST

The following table compares the two HTTP methods: GET and POST.

	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

HTTP – Responses

The server response the client as HTTP response message which contains the following information:

- A Status-line
- Zero or more header
- An empty line
- Optionally a message-body

The following sections explain each of the entities used in an HTTP response message.

Message Status-Line

A Status-Line consists of the protocol version followed by a numeric status code and its associated textual phrase. The elements are separated by space SP characters.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

HTTP Version

A server supporting HTTP version 1.1 will return the following version information:

HTTP-Version = HTTP/1.1

Status Code

The Status-Code element is a 3-digit integer where first digit of the Status-Code defines the class of response and the last two digits do not have any categorization role. There are 5 values for the first digit:

S.N.	Code and Description
1	1xx: Informational It means the request was received and the process is continuing.
2	2xx: Success It means the action was successfully received, understood, and accepted.
3	3xx: Redirection It means further action must be taken in order to complete the request.
4	4xx: Client Error It means the request contains incorrect syntax or cannot be fulfilled.
5	5xx: Server Error It means the server failed to fulfill an apparently valid request.

6.3 Life Cycle of Web Servlets

Three methods are central to the life cycle of a servlet. These are ***init( ), service( )***, and ***destroy( ).*** They are implemented by every *servlet* and are invoked at specific times by the server. Let us consider a typical user scenario to understand when these methods are called.

1. **First**, assume that a user enters a Uniform Resource Locator (URL) to a web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.

2. **Second**, the web server receives this HTTP request. The server maps this request to a particular ***servlet***. The servlet is dynamically retrieved and loaded into the address space of the server.
3. **Third**, the server invokes the ***init( )*** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.
4. **Fourth**, the server invokes the ***service( )*** method of the servlet. This method is called *to process the HTTP request.* It may also formulate an HTTP response for the client. The servlet remains in the *server's address space* and is available to process any other HTTP requests received from clients. The ***service( )*** method is called for each HTTP request.
5. **Finally**, the server may decide to ***unload*** the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the ***destroy( )*** method to relinquish any resources such as file handles that are allocated for the servlet. Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.

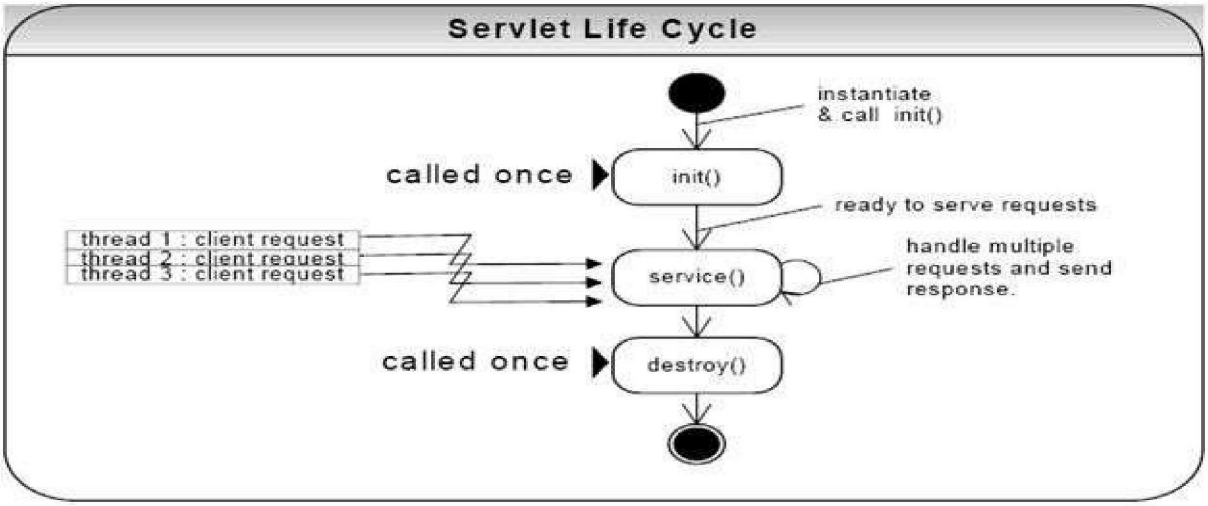


Fig4: Servlet Life cycle

6.4 Writing Servlet Programs with Servlet APIs

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write Servlets that extend **javax.servlet.http.HttpServlet**, an abstract class that implements the Servlets interface and is specially designed to handle HTTP requests.

```
// Import required java libraries
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloWorld extends HttpServlet{
      private String message;
      public void init() throws ServletException {
            // Do required initialization
            message ="Hello World";
      }
```

```
      public void doGet(HttpServletRequest request,HttpServletResponse response)
throws ServletException,IOException{
            // Set response content type
            response.setContentType("text/html");
            // Actual logic goes here.
            PrintWriter out= response.getWriter();
            out.println("<h1>"+ message +"</h1>");
      }
      public void destroy()   {
            // do nothing.
      }
}
```

Compiling a Servlets:

Let us put above code if HelloWorld.java file and put this file in C:\ServletDevel (Windows) or /usr/ServletDevel(Unix) then you would need to add these directories as well in CLASSPATH. Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows:

```
javac HelloWorld.java
```

Servlets Deployment:

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

```
<servlet>
      <servlet-name>HelloWorld</servlet-name>
      <servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>HelloWorld</servlet-name>
      <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

You are almost done, now let us start tomcat server using <Tomcat-installation-directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get following result:

6.5 Reading and Processing Forms

Reading Servlet Parameters

The ***ServletRequest*** interface includes methods that allow us to read the names and values of parameters that are included in a client request. The example contains two files. A web page is defined in ***EmployeeInfo.html***, and a servlet is defined in ***EmployeeServlet.java***. The HTML source code for ***EmployeeInfo.html*** is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

```
<html>
<head>
<title>Employee Info</title>
</head>
<body>
     <form name="EmployeeForm" method="post" action="EmployeeServlet">
          <table>
               <tr>
                    <td>Employee</td>
                    <td><input type="text" name="name"></td>
               </tr>
               <tr>
                    <td>Phone</td>
                    <td><input type="text" name="phone"></td>
               </tr>
          </table>
          <input type="submit" value="Submit">
     </form>
</body>
</html>
```

The source code for *EmployeeServlet.java* is shown in the following listing. The ***service***( ) method is overridden to process client requests. The ***getParameterNames***( ) method returns an enumeration of the parameter names. These are processed in a loop. We can see that the parameter name and value are output to the client. The parameter value is obtained via the ***getParameter***( ) method.

```
import java.io.*;
import java.util.*;
import jakarta.servlet.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/EmployeeServlet")
public class EmployeeServlet extends GenericServlet {
     public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException {

          // Get print writer.
          PrintWriter pw = response.getWriter();

          // Get enumeration of parameter names.
          Enumeration<String> e = request.getParameterNames();

          // Display parameter names and values.
          while(e.hasMoreElements()) {
               String pname = (String)e.nextElement();
               pw.print(pname + " = ");
               String pvalue = request.getParameter(pname);
               pw.println(pvalue);
          }
          pw.close();
     }
}
```

Compile the servlet. Next, copy it to the appropriate directory, and update the ***web.xml*** file. Then, perform these steps to test this example:

1. Start Tomcat (if it is not already running).
2. Display the web page in a browser.
3. Enter an employee name and phone number in the text fields.
4. Submit the web page.

After following these steps, the browser will display a response that is dynamically generated by the servlet.

6.6 Handling GET/POST Requests

Access Form Data or Parameter data using GET method and POST method

Servlets handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

1. GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the character as follows:

https://localhost:8080/hello?key1=value1&key2=value2

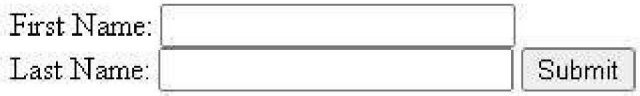
The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location box. Never use the GET method if you have password or other sensitive information to pass to the server. Servlets handles this type of requests using **doGet()**method.

A. GET Method Example Using Form:

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlets HelloForm.java to handle this input.

```
<html>
<body>
      <form action="HelloForm" method="GET">
            First Name: <input type="text" name="first_name"> <br /> Last
            Name: <input type="text" name="last_name" /> <input
type="submit"
                  value="Submit" />
      </form>
</body>
</html>
```

When you would access *http://localhost:8080/hello.html*, here is the actual output of the above form.



B. GET Method Example Using URL:

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name=Sunil&last_name=Bist

Below is **HelloForm.java** Servlets program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Extend HttpServlet class
public class HelloForm extends HttpServlet
{
     public void doGet(HttpServletRequest request,HttpServletResponse
response)  throws ServletException,IOException
     {
          // Set response content type
          response.setContentType("text/html");
          PrintWriter out= response.getWriter();
          String title ="Using GET Method to Read Form Data";
          String docType = "<! doctype html > ";
          out.println(docType + "<html>\n"+
                    "<head> <title>"+ title +"</title> </head>\n"+
                    "<body bgcolor=\"#f0f0f0\">\n"+
                    "<ul>\n"+
                    " <li><b>First Name</b>: " +
request.getParameter("first_name")+"\n"+
                    " <li><b>Last Name</b>: " +
request.getParameter("last_name")+"\n"+
                    "</ul>\n"+
                    "</body></html>");
     }
}
```

Assuming your environment is setup properly, compile HelloForm.java as follows:

javac HelloForm.java

Using GET Method to Read Form Data

First Name: Sunil

Last Name: Bist

2. POST method:

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlets handles this type of requests using **doPost()** method.

Hello.html file with the POST method as follows:

```
<html>
<body>
     <form action="HelloForm" method="POST">
          First Name: <input type="text" name="first_name"> <br /> Last
          Name:
          <inputtype ="text"name="last_name" />
          <input type="submit" value="Submit" />
     </form>
</body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name:
Last Name: Submit

Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

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



public class HelloForm extends HttpServlet
{
     // Method to handle GET method request.
     public void doGet(HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException
     {
          // Set response content type
          response.setContentType("text/html");
          PrintWriter out= response.getWriter();
          String title ="Using GET Method to Read Form Data";
          String docType = "<!doctype html ">\n";
                    out.println(docType + "<html>\n"+
```

```
                                    "<head> <title>"+ title +"</title>
</head>\n"+
                                    "<body bgcolor=\"#f0f0f0\">\n"+
                                    "<ul>\n"+
                                    "<li><b>First Name</b>: " +
request.getParameter("first_name")+"\n"+
                                    " <li><b>Last Name</b>: " +
request.getParameter("last_name")+"\n"+
                                    "</ul>\n"+
                                    "</body></html>");
      }
      // Method to handle POST method request.
      public void doPost(HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException
      {
            doGet(request, response);
      }
}
```

6.7 Database Connectivity through Servlets

Note: We Will cover all the CRUD functionalities in Lab Work

6.8 Cookies and Sessions

Cookies Handling

Cookies are text files stored on the client computer and they are kept for various information and tracking purpose. Java Servlets transparently supports HTTP cookies. There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example, name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A Servlets that sets a cookie might send headers that look something like this:

```
HTTP/1.1200 OK

Date:Fri,04 Feb200021:03:38 GMT

Server: Apache/1.3.9(UNIX) PHP/4.0b3

Set-Cookie: name=xyz; expires=Friday, 04-Feb-0722:03:38 GMT;
```

path=/; domain=tutorialspoint.com

Connection: close

Content-Type: text/html

The names and values of cookies are stored on the user's machine. Some of the information that is saved for each cookie includes the following:

- The name of the cookie
- The value of the cookie
- The expiration date of the cookie
- The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user's machine.

Servlets Cookies Methods:

S.N.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	**public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
5	**public String getName()** This method returns the name of the cookie. The name cannot be changed after creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie.
7	**public String getValue()** This method gets the value associated with the cookie.
8	**public void setPath(String uri)** This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	**public String getPath()** This method gets the path to which this cookie applies.

Setting Cookies with Servlets:

Setting cookies with Servlets involves three steps:

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

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

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

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

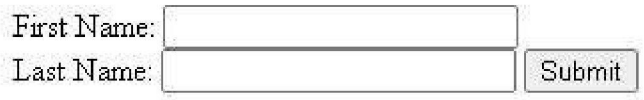
3. Sending the Cookie into the HTTP response headers: You use response.addCookie to add cookies in the HTTP response header as follows:

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

Cookies example using Servlets

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

```
<html>
<body>
      <form action="HelloForm" method="GET">
            First Name: <input type="text" name="first_name"> <br />
            Last Name: <input type="text" name="last_name" />
            <input type="submit" value="Submit" />
      </form>
</body>
</html>
```




```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
public class HelloForm extends HttpServlet{
      public void doGet(HttpServletRequest request,HttpServletResponse
response) throws ServletException,IOException
      {
            // Create cookies for first and last names.
            Cookie firstName =new Cookie("first_name",
request.getParameter("first_name"));
            Cookie lastName =new Cookie("last_name",
request.getParameter("last_name"));

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

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

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

            response.setContentType("text/html");
            PrintWriter out= response.getWriter();
            String title ="Setting Cookies Example";
            String docType ="<!doctype html>\n";
            out.println(docType + "<html>\n"+
                        "<head><title>"+ title +"</title></head>\n"+
                        "<body bgcolor=\"#f0f0f0\">\n"+
                        "<ul>\n"+
                        " <li><b>First Name</b>: " +
request.getParameter("first_name")+"\n"+
                        " <li><b>Last Name</b>: " +
request.getParameter("last_name")+"\n"+
                        "</ul>\n"+
                        "</body></html>");
      }
}
```

Try to enter First Name and Last Name and then click submit button. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

Reading Cookies with Servlets:

To read cookies, you need to create an array of javax.servlet.http.Cookie objects by calling the **getCookies( )** method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

Example:

```
import java.io.*;

import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class ReadCookies extends HttpServlet{
      public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException,IOException {

            Cookie cookie =null;
            Cookie[] cookies =null;

            // Get an array of Cookies associated with this domain
```

```
        cookies = request.getCookies();

        // Set response content type
        response.setContentType("text/html");
        PrintWriter out= response.getWriter();
        String title ="Reading Cookies Example";
        String docType = "<!doctype html>\n";
        out.println(docType + "<html>\n"+
                    "<head><title>"+ title +"</title></head>\n"+
                    "<body bgcolor=\"#f0f0f0\">\n");
        if( cookies !=null){
             out.println("<h2> Found Cookies Name and Value</h2>");
             for(int i =0; i < cookies.length; i++){
                  cookie = cookies[i];
                  out.print("Name : "+ cookie.getName()+", ");
                  out.print("Value: "+ cookie.getValue()+" <br/>");
             }
        }
        else
        {
             out.println( "<h2>No cookies founds</h2>");
        }
        out.println("</body>");
        out.println("</html>");
    }
}
```

Found Cookies Name and Value

Name : first_name, Value: Sunil Name : last_name, Value: Bist

Delete Cookies with Servlets:

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps:

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using setMaxAge() method to delete an existing cookie.
- Add this cookie back into response header.

Session Handling in Java Servlets

The HttpSession interface enables a Servlets to read and write the state information that is associated with an HTTP session. Several of its methods are summarized in Table below. All of these methods throw an IllegalStateException if the session has already been invalidated.

Method	Description
Object getAttribute(String *attr*)	Returns the value associated with the name passed in *attr*. Returns **null** if *attr* is not found.
Enumeration getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when this session was created.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request for this session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns **true** if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String *attr*)	Removes the attribute specified by *attr* from the session.
void setAttribute(String *attr*, Object *val*)	Associates the value passed in *val* with the attribute name passed in *attr*.

Session Example in Java Servlets

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

import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class SessionTrack extends HttpServlet{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException  {

      // Create a session object if it is already not created.
      HttpSession session = request.getSession(true);

      // Get session creation time.
      Date createTime =new Date(session.getCreationTime());

      // Get last access time of this web page.
      Date lastAccessTime = new Date(session.getLastAccessedTime());

      String title ="Welcome Back to my website";
      Integer visitCount =new Integer(0);
      String visitCountKey =new String("visitCount");
      String userIDKey =new String("userID");
      String userID =new String("ABCD");

      // Check if this is new comer on your web page.
      if(session.isNew())   {
            title ="Welcome to my website";
            session.setAttribute(userIDKey, userID);
```

```
        }
        else {
              visitCount =(Integer)session.getAttribute(visitCountKey);
              visitCount = visitCount +1;
              userID =(String)session.getAttribute(userIDKey);
        }
        session.setAttribute(visitCountKey, visitCount);
        // Set response content type
        response.setContentType("text/html");
        PrintWriter out= response.getWriter();

        String docType = "<!doctype html>\n";
        out.println(docType +
                    "<html>\n"+
                    "<head><title>"+ title +"</title></head>\n"+
                    "<body bgcolor=\"#f0f0f0\">\n"+ "<table>"+
                    " <td>"+ session.getId()+"</td></tr>\n"+
                    "<tr>\n"+
                    " <td>Creation Time</td>\n"+
                    " <td>"+ createTime +
                    " </td></tr>\n"+
                    "<tr>\n"+
                    " <td>Time of Last Access</td>\n"+
                    " <td>"+ lastAccessTime +
                    " </td></tr>\n"+
                    "<tr>\n"+
                    " <td>User ID</td>\n"+
                    " <td>"+ userID +
                    " </td></tr>\n"+
                    "<tr>\n"+
                    " <td>Number of visits</td>\n"+
                    " <td>"+ visitCount +"</td></tr>\n"+
                    "</table>\n"+
                    "</body></html>");
}
}
```

Deleting Session Data:

<session-config>

<session-timeout>15</session-timeout>

</session-config>