Java Beans

JavaBeans are classes that encapsulate many objects into a single object (the bean). It is a java class that should follow following conventions:

1- Java Bean must implement Serializable (Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object)

2- It should have a public no-arg constructor.

3- All properties in java bean must be private with public getters and setter methods.

```
public class TestBean {
        private String name;
        public void setName(String name) {
                this.name = name;
        }
        public String getName() {
        return name;
        }
        public boolean isempty() {
                return empty;
        }
}
```

Syntax for setter methods

It should be public in nature and the return-type should be void.
The setter method should be prefixed with set.
It should take some argument i.e. it should not be no-arg method.

Syntax for getter methods

1- It should be public in nature and the return-type should not be void.
2- The getter method should be prefixed with get.
3- It should not take any argument.
4- For Boolean properties getter method name can be prefixed with either "get" or "is". But recommended to use "is".

Components of JavaBeans

The classes that contained definition of beans is known as components of JavaBeans. These classes follows certain design conventions. It includes properties, events, methods and persistence. There are two types of components, GUI based and non GUI based. For instance JButton is example of a component not a class.

Properties (date members): Property is a named attribute of a bean, it includes color, label, font, font size, display size. It determines appearance, behavior and state of a bean.
Methods: Methods in JavaBeans are same as normal Java methods in a class. All properties should have accessor and getter methods.
Events: Events in JavaBeans are same as SWING/AWT event handling.
Persistence: Serializable interface enables JavaBean to store its state.

Advantages of JavaBeans

1- Reusability in different environments. Can be deployed in network systems
2- Used to create applet, servlet, application or other components.
3- JavaBeans are dynamic, can be customized.
4- The properties, events, and methods of a bean can be exposed to another application.
5- A bean may register to receive events from other objects and can generate events that are sent to those other objects.
6- Auxiliary software can be provided to help configure a bean.
7- The configuration settings of a bean can be saved to persistent storage and restored.

Server Side Programming

Writing programs to create dynamic pages is called server side programming since the programs run on the web server. Server-side programming must deal with dynamic contents and includes following operations.

The following are the different types of server side programming languages

1- Java Servlets
2- Java Server Pages(JSP)
3- Active Server Pages (ASP)
4-Hypertext Preprocessor (PHP)

Server Side Programming Performs Following Tasks

1. Querying the database/Operations over databases
2. Access/Write a file on server.
3. Interact with other servers.
4. Structure web applications.
5. Process user input.

Advantages of Server Side Programs

Some of the advantages of Server Side programs are as follows:

1- All programs reside in one machine called server machine. Any number of remote machines (called client machines) can access the server programs.

2- New functionalities to existing programs can be added at the server side which the clients can take advantage of without having to change anything.

3- Migrating to newer versions, architectures, design patterns, switching to new databases can be done at the server side without having to bother about client's hardware or software capabilities.

4- Issues relating to enterprise applications like resource management, concurrency, session management, security and performance are managed by the server side applications.

5- They are portable and possess the capability to generate dynamic and user-based content (e.g., displaying transaction information of credit card or debit card depending on user's choice).

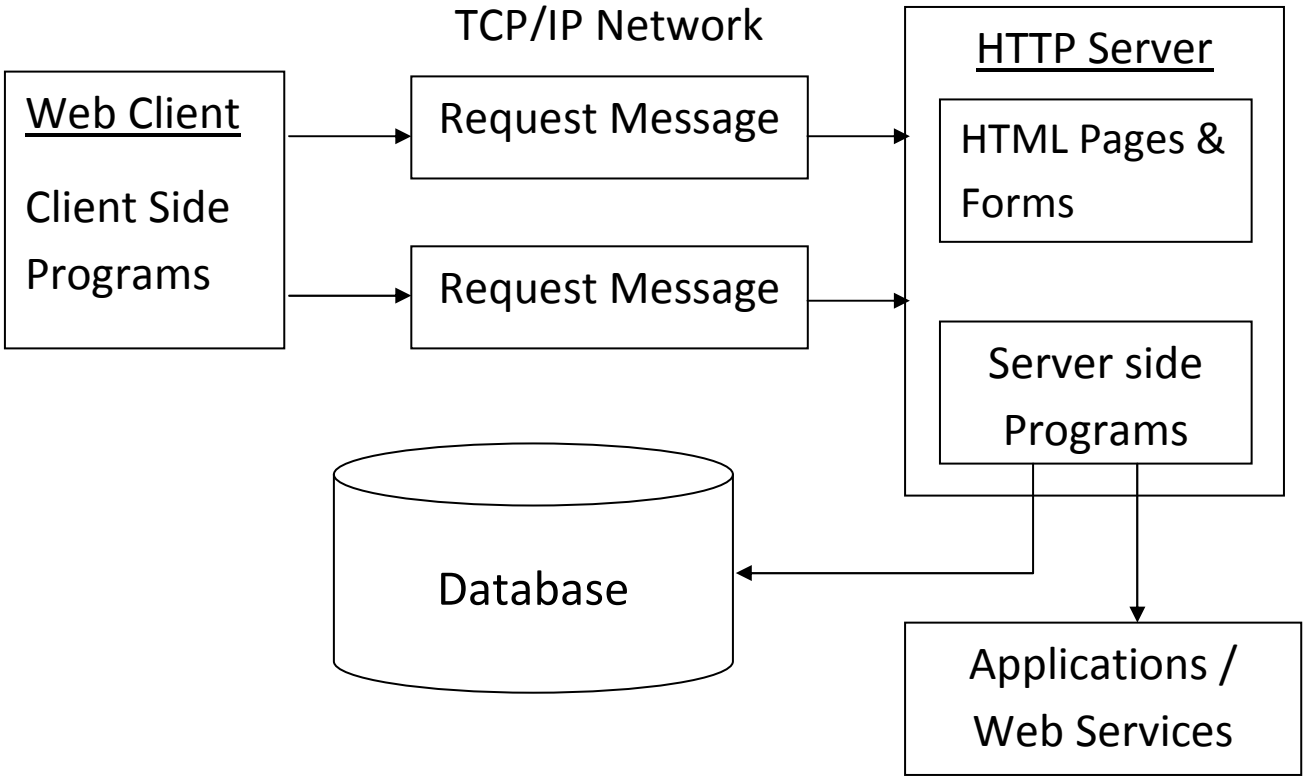
Servlet

Servlet is a **Single Instance** and **Multiple threads** principle base server side technology to develop server side components (A reusable java object is called as java "component") as web resource programs of web applications.

When your web server (like Apache) gets a request for a servlet from the client, the server hands over the request not to the servlet itself, but to the servlet container in which servlet is deployed. The servlet container then directs the request to the appropriate servlet.

The servlet does its processing which may include interacting with the database or other server side components such as servlets or JSPs (Java Server Pages). After the request is processed by servlet, the response

(generally in the form of HTML Document) is returned back to the servlet container which in turn sends the response back to the client via the web server.



Servlets are generally used for

1- Processing and storing data submitted using HTML form by the user such as purchase order or a credit card data.

2- Providing dynamic contents on web pages (Returning the results of a database query to the client).

3- Allowing collaboration between people. A servlet can handle multiple requests concurrently (Servlet can synchronize requests to support systems such as online conferencing).

4- Forwarding requests to other servers and servlets to balance load among several servers that have the same contents.

5- Managing state information on the top of the stateless HTTP e.g. for an online shopping cart system which manages shopping carts from many concurrent customers and maps every request to the right customer.

Servlet Life Cycle

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the init() method.
- The servlet calls service() method to process a client's request.
- The servlet is terminated by calling the destroy() method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations

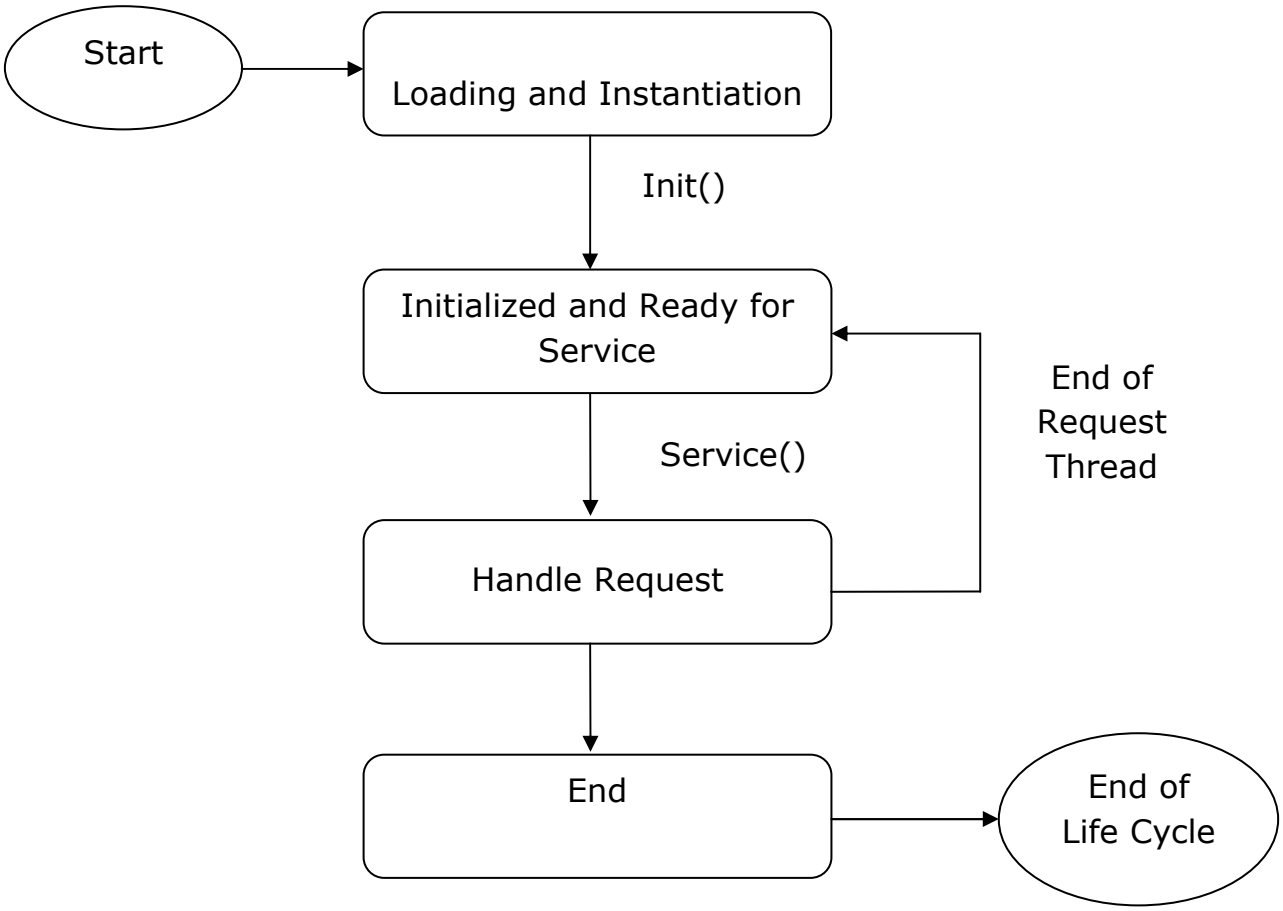


Fig: Servlet Life Cycle Flow Chart

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client (web browsers) and response back to the client.

Each time the server receives a request for a servlet, the server creates a new thread and calls service. The service() method checks the HTTP request type (GET, POST) and calls doGet, doPost methods as appropriate.

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method used to close database connections, halt background threads, write cookie lists and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection.

Http Servlet Example

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

public class MyExamServlet extends HttpServlet
{
	private String message;

	public void init() throws ServletException
	{
		message = "UBTER Semester Exam 2019";
	}

	public void doGet(HttpServletRequest request,
				HttpServletResponse response)
				throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		out.println("<h1>" + message + "</h1>");

	}
```

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

GET & POST Request

HTTP Protocol make request for resource using GET and POST methods. HTTP POST requests supply additional data from the client to the server in the message body in secure way whereas GET request supply data in URL in efficient way. Servlet Handles GET and POST request using its doGet() and doPost() methods.

The GET Method

GET is used to request data from a specified resource. GET is one of the most common HTTP methods. Note that the query string (name/value pairs) is sent in the URL of a GET request:

../LeExamTest/RegForm ? name=Ram & Add=DDN

1. GET requests can be cached
2. GET requests remain in the browser history
3. GET requests can be bookmarked
4. GET requests should never be used when dealing with sensitive data
5. GET requests have length restrictions
6. GET requests is only used to request data (not modify)

The POST Method

POST is used to send data to a server to create/update a resource. POST is one of the most common HTTP methods. The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST  /test/RegForm.jsp HTTP/1.1
Host   : localhost:8080
name1=value1&name2=value2
```

1. POST requests are never cached
2. POST requests do not remain in the browser history
3. POST requests cannot be bookmarked
4. POST requests have no restrictions on data length

Methods doGet() versus doPost() Methods

doGet() Method	doPost() Method
Method doGet is called when a HTTP GET request is made. doGet() is called when users click on a link, or enter a URL.	Method doPost is called is when a HTTP POST request is made. doGet() is called when
It also happens with some HTML FORMs (those with METHOD="GET" specified in the FORM tag).	This also happens with HTML FORMs (those with METHOD="POST" specified in the FORM tag).
In doGet Method the parameters are appended to the URL and sent along with header information.	In doPost method , form data is sent in separate line in the body.
Maximum size of data that can be sent is limited. Parameters sent are not encrypted.	Data that can be sent is not limited. Parameters are sent in encrypted form.
doGet is faster if we set the response content length since the same connection is used.	doPost is slower compared to doGet since doPost does not write the content response length.
GET Request can be cached, can be bookmarked, have restriction on length of data, and can be retain in the browser history. Only ASCII data is allowed.	POST Request cannot be bookmarked, No restriction on length of data, never cached, and not retain in the browser history. Binary data also allowed.

Configuration of Tomcat Server

1- Configure Java_Home and User for the Tomcat- Open the startup.bat file located in the bin folder of the Tomcat.

set JAVA_HOME="C:\DevPrograms\Java\jdk1.8.0_144"

2- Configure the users that are allowed to use Tomcat. Open the tomcat-users.xml file

Tomcat have defined the following four roles in advance:

1- manager-gui - allows access to the HTML GUI and the status pages
2- manager-script - allows access to the text interface and the status pages
3- manager-jmx - allows access to the JMX proxy and the status pages
4- manager-status - allows access to the status pages only

User can have one or more roles. Now I will declare a user named " tomcat" and having 4 above roles.

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<role rolename="manager-status"/>

<user username="gplohaghat" password="mypassword"
    roles="manager-gui, manager-script, manager-jmx, manager-status"/>
```

3- Run the Tomcat- To run the Tomcat double-click on startup.bat in the bin folder of the Tomcat.

4- In the browser, visit the address: http://localhost:8080/

5- Click on Manager App Button and Input User Name and Password and click on Login Button Or visit the address: http://localhost:8080/Manager/html.

6- Deploy the application onto the Tomcat- First of all, Create an Application MyWebApp and deploy it onto the Tomcat by Setting following fields under Deploy Section and then Click on Deploy Button.

Context Path(Require) : /MyWebApp
WAR or Directory URL : C:/MyApps/MyWebApp.war

7- Run the demo application- In the browser, visit the address: http://localhost:8080/ MyWebApp.

8- Open the server.xml file located in config folder and Change Port 8080 to 80 and Timeout to 12000.
<Connectorport = "80" protocol = "HTTP/1.1"
connectionTimeout = "120000" redirectPort = "8443">

9- Sometimes during deployment an application onto the Tomcat, but UTF-8 doesn't work. You can configure the UTF-8 as the default charset for the Tomcat.

set JAVA_OPTS=-Djavax.servlet.request.encoding=UTF-8 -Dfile.encoding=UTF-8

CGI versus Servlet

The Common Gateway Interface (CGI) is the first technology used to generate dynamic contents. It allows a web client to pass data to the application running on the web server so that a dynamic web page can be returned to the client according to the input data.
CGI is not a programming language, rather it is an interface (or a set of rules) that allows an input from a web browser and produce an output in the form of HTML page. A CGI script can be written in a verity of languages such as C, C++, Visual Basic, Perl, FORTRAN and even Java. Out of these, Perl is most commonly used.
When a web server receives a request for a CGI script, the web server passes some parameters to this script and executes it. The script runs and generates some output which is then collected by the web server and returned to the client (browser).

Advantages of CGI

1. CGI is a true cross-platform technology. This means that CGI scripts work with any web browser as well as with most web servers running on Windows and Unix.

2. CGI is language independent. It can be written in a variety of languages so developers do not have to learn a new language.

3. CGI is very simple interface. It is not necessary to have any special libraries to create a CGI program or write programs using a particular API.

Disadvantages of CGI

1. A new process is started for each client request. The creation of the process for every such request requires time and significant server resources which limits the number of requests a server can handle.

2. CGI program cannot interact with the web server or take advantage of the server's abilities once it begins execution. This is because it is running in a separate process. For example, a CGI script cannot write to the server's log file.

Difference between SERVLET and CGI

Both Java servlets and CGI are used for creating dynamic web applications that accept a user request, process it on the server side and return responses to the user. However, Java servlets provide a number of advantages over traditional CGI which are as follows,

Efficient: Unlike traditional CGI where a new process is started for each client request, a servlet processes each request as a thread inside of a process.
CGI program which terminates after handling a request, the servlets remains in memory thus servlets make it easier to cache computations, keep database connections open.

Powerful: Servlets support several capabilities that are difficult or impossible to accomplish with traditional CGI. These capabilities include talking directly to the web server, sharing data between multiple servlets, session tracking and caching of previous computations.

Portable: Since servlets are written in the Java programming language and follow a standard API, so a servlet can be moved from one servlet compatible web server to another very easily.

Inexpensive: A number of free or very inexpensive web servers are available these days. Once you have a web server, adding servlet support to it costs very little.

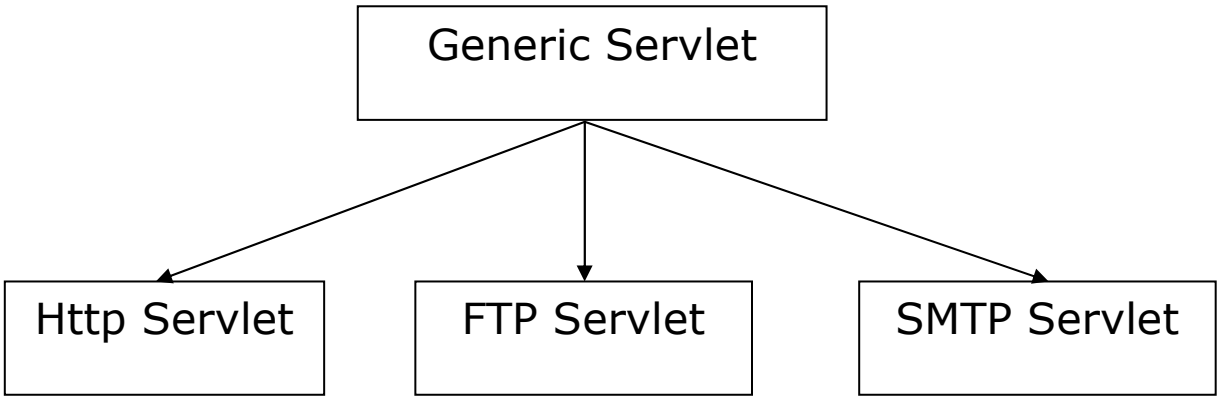
Language dependency: Since servlets can be written only in Java so they are language dependent. On the other hand, CGI programs can be written in any programming language like C, C++, Perl.

Secure: Servlet can be run by a servlet engine or servlet container which increases the server security. On the other hand, CGI scripts are significantly less secure.

Convenient: Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, session handling etc. On the other hand, CGI does not support such infrastructure.

Type of Servlets

There is a possibility of developing 'n' types of servlets, like httpservlet, ftpservlet, smtpservlet etc. for all these protocol specific servlet classes GenericServlet is the common super class containing common properties and logics. So, GenericServlet is not a separate type of servlet.



Generic servlets extend javax.servlet.GenericServlet- It is protocol independent servlet. Generic Servlet is a base class servlet from which all other Servlets are derived. Generic Servlet supports for HTTP, FTP and SMTP

protocols. It implements the Servlet and ServletConfig interface. It has only init() and destroy() method of ServletConfig interface in its life cycle.

HTTP servlets extend javax.servlet.HttpServlet- HTTPServlet is HTTP dependent servlet. The HTTP protocol is a set of rules that allows Web browsers and servers to communicate. When Web browsers and servers support the HTTP protocol, Java-based web applications are dependent on HTTP Servlets.HttpServlet is Extended by Generic Servlet.

HttpServlet Supports Following HTTP Methods

1- GET
2- PUT
3- DELETE
4- OPTIONS
5- POST
6- HEAD
7- PATCH

HttpServlet

HttpServlet is an abstract class given under the servlet-api present. It is present in javax.servlet.http package and has no abstract methods. It extends GenericServlet class.
When the servlet container uses HTTP protocol to send request, then it creates HttpServletRequest and HttpServletResponse objects. HttpServletRequest binds the request information like header and request methods and HttpServletResponse binds all information of HTTP protocol.

Methods in HttpServlet: The methods in httpservlet are given as follows:

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

Called by the server (via the service method) to allow a servlet to handle a GET request. The HTTP GET method allows the client to send limited size of data.

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

Called by the server (via the service method) to allow a servlet to handle a POST request. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.

protected void doDelete (HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

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.

protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException

Receives standard HTTP requests from the public service method and dispatches them to the doXXX methods defined in this class. This method is an HTTP specific version of the service method. There's no need to override this method.

public void service (ServletRequest req/ ServletRespon,se res) throws ServletException, IOException

Dispatches client request to the protected service method. There's no need to override this method.

protected long getLastModified (HttpServletRequest req)

Returns the time the HttpServletRequest object was last modified, in milliseconds.

Servlet Container/Web container

Servlet container (also known as a Web container) is a software or software application that Servlet container takes care of servlet program life cycle (Object birth to death) of given resource (like java classes) and generates dynamic web pages. So servlet container is the essential part of the web server that interacts with the java servlets and communicates between client Browsers and the servlets.
There are a lot of Servlet Containers like Jboss, Apache Tomcat, WebLogic etc.

How does this Servlet Container work?

1- A client browser accesses a Web server or HTTP server for a page.

2- The Web server redirects the request to the servlet container (Servlets are HTTP listeners that run inside the servlet container) and the servlet container redirects the request to the appropriate servlet.

3- The servlet is dynamically retrieved and loaded into the address space of the container, if it is not in the container.

4- The servlet container invokes servlet init () method once when the servlet is loaded first time for initialization.

5- The servlet container invokes the service () methods of the servlet to process the HTTP request, i.e., read data in the request and formulate a response. The servlet remains in the container's address space and can process other HTTP requests.

6- Web servlet generates data (HTML page, picture ...) return the dynamically generated results to the correct location.

Responsibilities of Web Server

1- Listens to client request continuously (HTTP Request) and passes the HTTP request to an appropriate web resource program of web application (deployed web application).

2- Provides container software to execute server side programs (web resource programs) and gathers output generated by web resource programs.

3- Passes output of web resource programs to browser window as http response in the form of web page.

4- Provide environment to deploy manage and to undeploy the web application.

How many ways we can develop a servlet?

The three important resources of servlet API.

1. javax.servlet.Servlet
2. javax.servlet.GenericServlet (Abstract class)
3. javax.servlet.http.httpServlet (Abstract class)

Every servlet program is a java class that is developed based on servlet api. There are three ways to develop servlet program.

1. Take a java class implementing javax.servlet.Servlet interface and provide implementation for all the five methods of that interface.

2. Take java class extending from javax.servlet.GenericServlet class and provide implementation for service() method.

3. Take java class extending from javax.servlet.http.httpServlet class and override one of the seven doxxx() methods or one of the two service(- -) methods.

In the above said three approaches the programmer overrides the service() and doXXX(-) methods and places request processing logic inside method body that generates web pages by processing the request. But programmer never calls these methods manually but Servlet container calls these methods automatically for every request given by client to servlet program.

Programmer just supplies his servlet program and related java class to servlet container then onwards servlet container is responsible to manage the whole life cycle of servlet program. Servlet program uses request object to read details from the request and response object to send response content to browser window through web server.

For Example when 10 requests are given to a servlet program from single or different browser windows (clients) then Servlet container creates 10 threads on servlet program objects representing 10 requests. Servlet container creates 10 sets of request, response objects and calls service (-,-) or methods for 10 times having request, response objects as arguments values.

Handling Http Request & Responses

GET Method Example using URL:- Here is a simple URL which will pass two values to RegistrationForm program using GET method.
http://localhost:8080/RegistrationForm?first_name = UBTER & last_name = ROORKEE

Given below is the RegistrationForm.java servlet 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 java.io.*;
import javax.servlet.*;
```

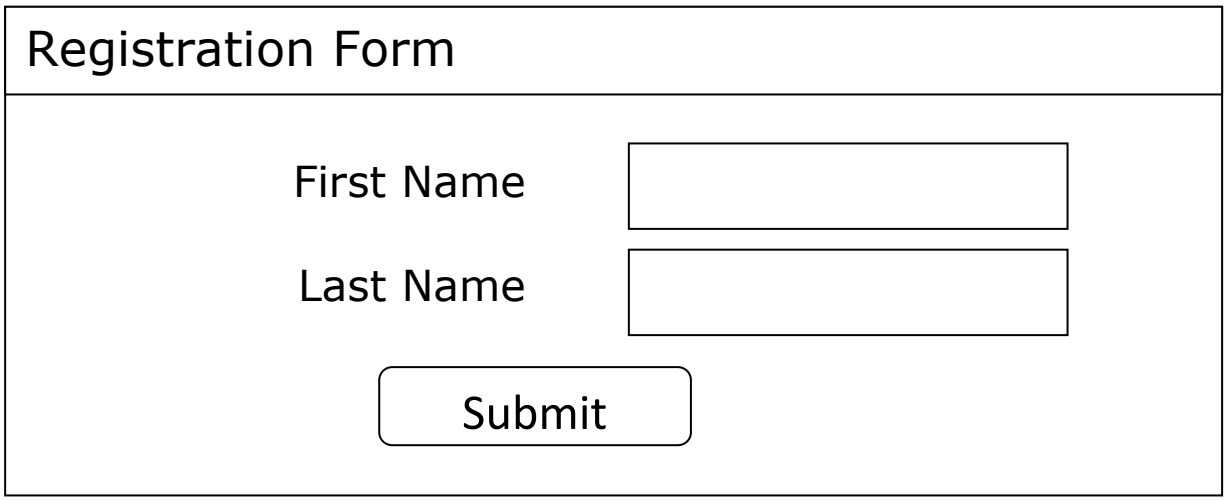
```
import javax.servlet.http.*;

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

     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String title = "GET Method to Read Form Data";

     String docType = "<!doctype html public \"-//w3c//dtd html
                                4.0 " + "transitional//en\">\n";

     out.println(docType +  "<html>\n" +
          "<head><title>" + title + "</title></head>\n" +
          "<body bgcolor = \"#f0f5e7\">\n" +
            "<h1 align = \"center\">" + title + "</h1>\n" +
            "<ul>\n" +
              "  <li><b>First Name</b>: "
              + request.getParameter("first_name") + "\n" +
              "  <li><b>Last Name</b>: "
              + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
          "</body>" + "</html>" );
  }
}
```

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 Servlet HelloForm to handle this input.

```
<html>
  <body>
    <form action = "RegistrationForm" 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>
```

Keep this HTML in a file Registration.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access http://localhost:8080/ Registration.htm, here is the actual output of the above form.



Enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form:- Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

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

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

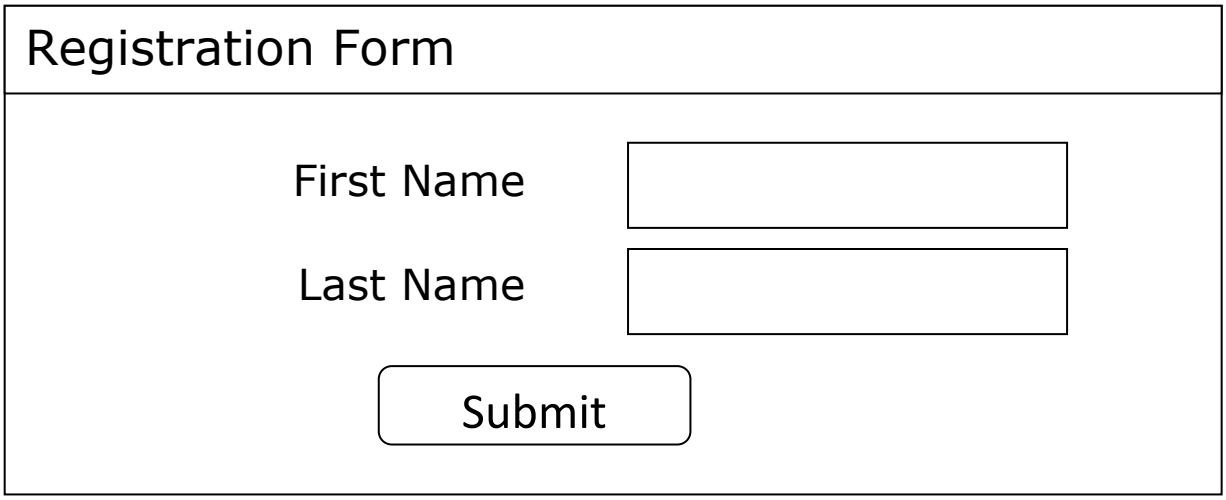
```
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String title = "GET Method to Read Form Data";

      String docType = "<!doctype html public \"-//w3c//dtd html
                                4.0 " + "transitional//en\">\n";

        out.println(docType +  "<html>\n" +
          "<head><title>" + title + "</title></head>\n" +
          "<body bgcolor = \"#f0f5e7\">\n" +
            "<h1 align = \"center\">" + title + "</h1>\n" +
            "<ul>\n" +
              "  <li><b>First Name</b>: "
              + request.getParameter("first_name") + "\n" +
              "  <li><b>Last Name</b>: "
              + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
          "</body>" + "</html>" );



        // Method to handle POST method request.
        public void doPost(HttpServletRequest request,
                                HttpServletResponse response)
                                throws ServletException, IOException {

        doGet(request, response);     //Call doGet Method

    }
}
```

Now compile and deploy the above Servlet and test it using RegistrationForm.htm with the POST method as follows −

```
<html><body>
     <form action = "RegistrationForm" method = "POST">
        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>
```

Keep this HTML in a file Registration.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory.

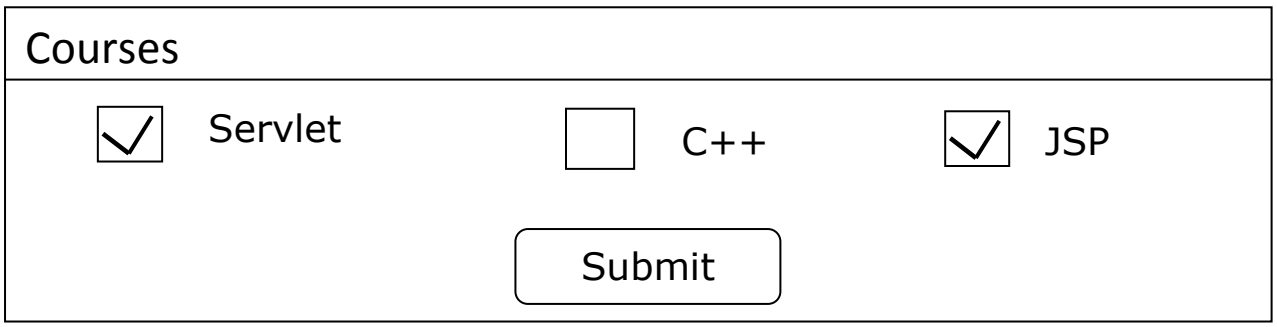


When you would access http://localhost:8080/ Registration.htm, here is the actual output of the above form. Enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running.

Passing Checkbox Data to Servlet Program:-Checkboxes are used when more than one option is required to be selected. Here is example HTML code, Courses.htm, for a form with two checkboxes

```
<html>
<body>
<form action = " Courses" method = "POST" target = "_blank">
        <input type = "checkbox" name = "Servlet"
                                        checked = "checked" /> Servlet
        <input type = "checkbox" name = "C++" /> C++
        <input type = "checkbox" name = "JSP"
                                        checked = "checked" /> JSP
        <input type = "submit" value = "Select Course" />
</form>
</body>
</html>
```

The result of this HTML code is the following HTML form.



Given below is the Courses.java servlet program to handle input given by web browser for checkbox button.

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

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

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading Checkbox Data";

        String docType = "<!doctype html public \"-//w3c//dtd html
                                4.0 " + "transitional//en\">\n";

        out.println(docType + "<html>\n" +
              "<head> <title>" + title + "</title></head>\n" +
              "<body bgcolor = \"#fdf2f3\">\n" +
            "<h1 align = \"center\">" + title + "</h1>\n" +
            "<ul>\n" + "<li><b>Course1 : </b>: "
                    + request.getParameter("Servlet") + "\n" +
                    "  <li><b>Course2: </b>: "
                          + request.getParameter("C++") + "\n" +
                    "  <li><b>Course3: </b>: "
```

```
                        + request.getParameter("JSP") + "\n" +
            "</ul>\n" +
              "</body>" + "</html>" );
  }
  public void doPost(HttpServletRequest request,
                                  HttpServletResponse response)
                                  throws ServletException, IOException {
        doGet(request, response);
  }
}
```

For the above example, it would display following result −

```
Servlet      : on
C++          : null
JSP          : on
```

Setting Cookies with Servlet

Setting cookies with servlet involves three steps −

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

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

Keep in mind, neither the name nor the value should contain white space or any of the following characters −
[ ] ( ) = , " / ? @ : ;

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

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

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

response.addCookie(cookie);

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

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

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

      // Create cookies for first and last names.
      FName = request.getParameter("first_name");
      LName = request.getParameter("last_name");

      Cookie firstName = new Cookie("first_name", FName);
      Cookie lastName = new Cookie("last_name", LName);

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

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

    // Set response content type
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Setting Cookies Example";
      out.println("<html><head><title>" + title +
                                "</title> + "</head>\n" +
```

```
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
          "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
          "</ul>\n" + "</body></html>");
  }
}
```

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

```
<html> <body>
    <form action = "StoreCookies" 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>
```

Keep above HTML content in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access http://localhost:8080/Hello.htm.

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.

Next section would explain you how you would access these cookies back in your web application.

Reading Cookies with Servlet:

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
Let us read cookies which we have set in previous example −

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

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

            Cookie cookie = null;
            Cookie[] cookies = request.getCookies();

            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String title = "Reading Cookies Example";


            out.println("<html> \n <head> \n <title>" + title +
                        "</title> \n </head>\n" +
                        "<body bgcolor = \"#fdf6e7\">\n");

            if( cookies != null ) {
                  out.println("<h2> Found Cookies</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>");
    }
}
```

Compile above servlet ReadCookies and create appropriate entry in web.xml file. If you would have set first_name cookie as "**Lohaghat**" and last_name cookie as "**Champawat**" then running http://localhost:8080/ReadCookies would display the following result-

Found Cookies
Name : first_name, Value: **Lohaghat**
Name : last_name, Value: **Champawat**

Delete Cookies with Servlet

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

1- Read an already existing cookie and store it in Cookie object.

2- Set cookie age as zero using setMaxAge() method to delete an existing cookie

3- Add this cookie back into response header.

Example
The following example would delete and existing cookie named "first_name" and when you would run ReadCookies servlet next time it would return null value for first_name.

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

// Extend HttpServlet class
public class DeleteCookies extends HttpServlet
{
```

```
   public void doGet(HttpServletRequest request,
                                HttpServletResponse response)
                                throws ServletException, IOException {
            Cookie cookie = null;    Cookie[] cookies = null;
            cookies = request.getCookies();
            response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Delete Cookies";

      out.println( "<html>\n" +
                  "<head><title>" + title + "</title></head>\n" +
                  "<body bgcolor = \"#f0f0f0\">\n" );

      if( cookies != null ) {
            out.println("<h2> Cookies Value</h2>");
            for (int i = 0; i < cookies.length; i++)
            {
                  cookie = cookies[i]
                  string cookieName = cookie.getName();

                  if(cookieName.compareTo("first_name") == 0 )
                  {
                  cookie.setMaxAge(0);
                  response.addCookie(cookie);
                  out.print("Deleted Cookie:");
                        out.print("cookie.getName());
                        out.print("<br/>");
            }
            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>");
  }
}
```

Compile above servlet DeleteCookies and create appropriate entry in web.xml file. Now running http://localhost:8080/DeleteCookies would display the following result –

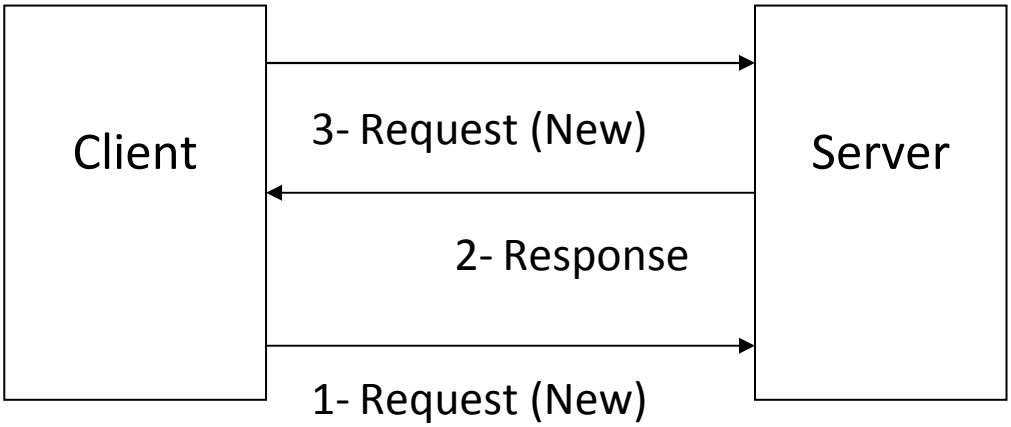
Cookies Value
Deleted cookie : first_name
Name : first_name, Value: Lohaghat
Name : last_name, Value: Champawat

Now try to run http://localhost:8080/ReadCookies and it would display only one cookie as follows –

Cookies Value
Name : last_name, Value: Champawat

Session Tracking in Servlet

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in Servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.



There are four techniques used in Session tracking:

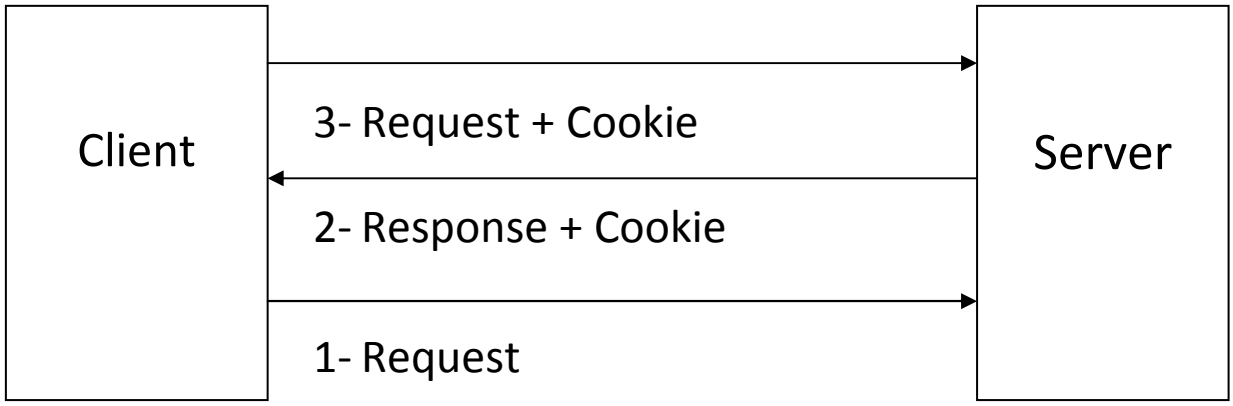
1- Cookies
2- Hidden Form Field
3- URL Rewriting
4- HttpSession

Cookies in Servlet

A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



There are 2 types of cookies in Servlets Persistent and Non-Persistent.

1- **Non-persistent cookie:** It is valid for single session only. It is removed each time when user closes the browser.
2- **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 or age of Cookie is expired.

Advantage of Cookies

1- Simplest technique of maintaining the state.
2- Cookies are maintained at client side.

Disadvantage of Cookies

1- It will not work if cookie is disabled from the browser.
2- Only textual information can be set in Cookie object.

Hidden Form Field

Hidden Form Field is a hidden (invisible) textfield that 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 (Every page is a form and have hidden form field) and we don't want to depend on the browser.

Let's see the code to store value in hidden field. Here, uname is the hidden field name and **GP Lohaghat** is the hidden field value.

```
<input type="hidden" name="uname" value="GP Lohaghat">
```

Real application of hidden form field

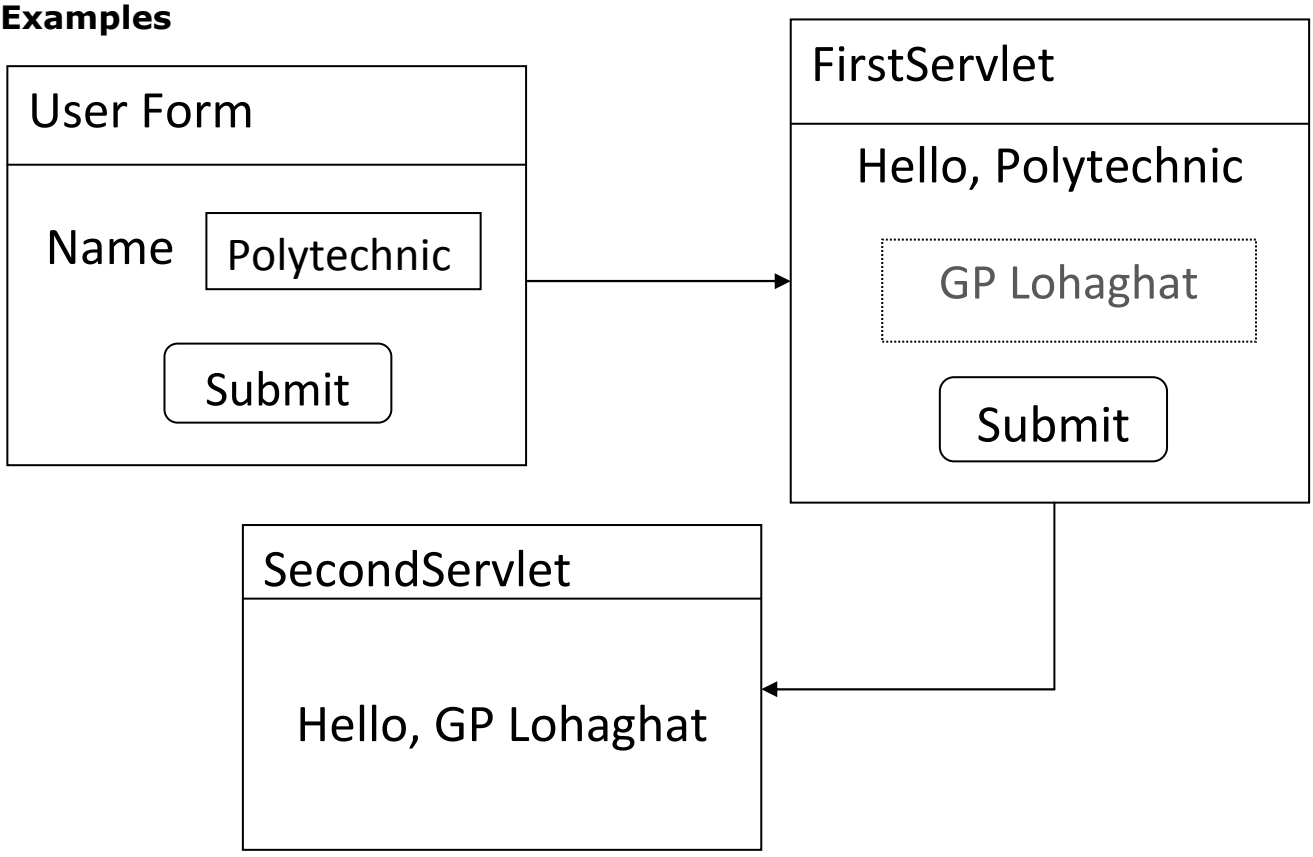
It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage: It will always work whether cookie is disabled or not.

Disadvantages

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

Examples



Code: HTML Form

```
<form action="FirstServlet">
        Name:<input type="text" name="userName"/><br/>
        <input type="submit" value="go"/>
</form>
```

Code:- FirstServlet.java

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

public class FirstServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request,
                                HttpServletResponse response) {

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

        String uName=request.getParameter("userName");
        out.print("Welcome "+n);
        out.print("<form action= 'SecondServlet'>");
        out.print("<input type='hidden' " +
                        name='uname' value='"+ uName +"'>");
        out.print("<input type='submit' value='go'>");
        out.print("</form>");
        out.close();
  }
  catch(Exception e) {
              System.out.println(e);
  }
}
```

Code:- SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                                    HttpServletResponse response)
        try
        {
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String n=request.getParameter("uname");
                out.print("Hello "+n);

                out.close();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
  }
}
```

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.

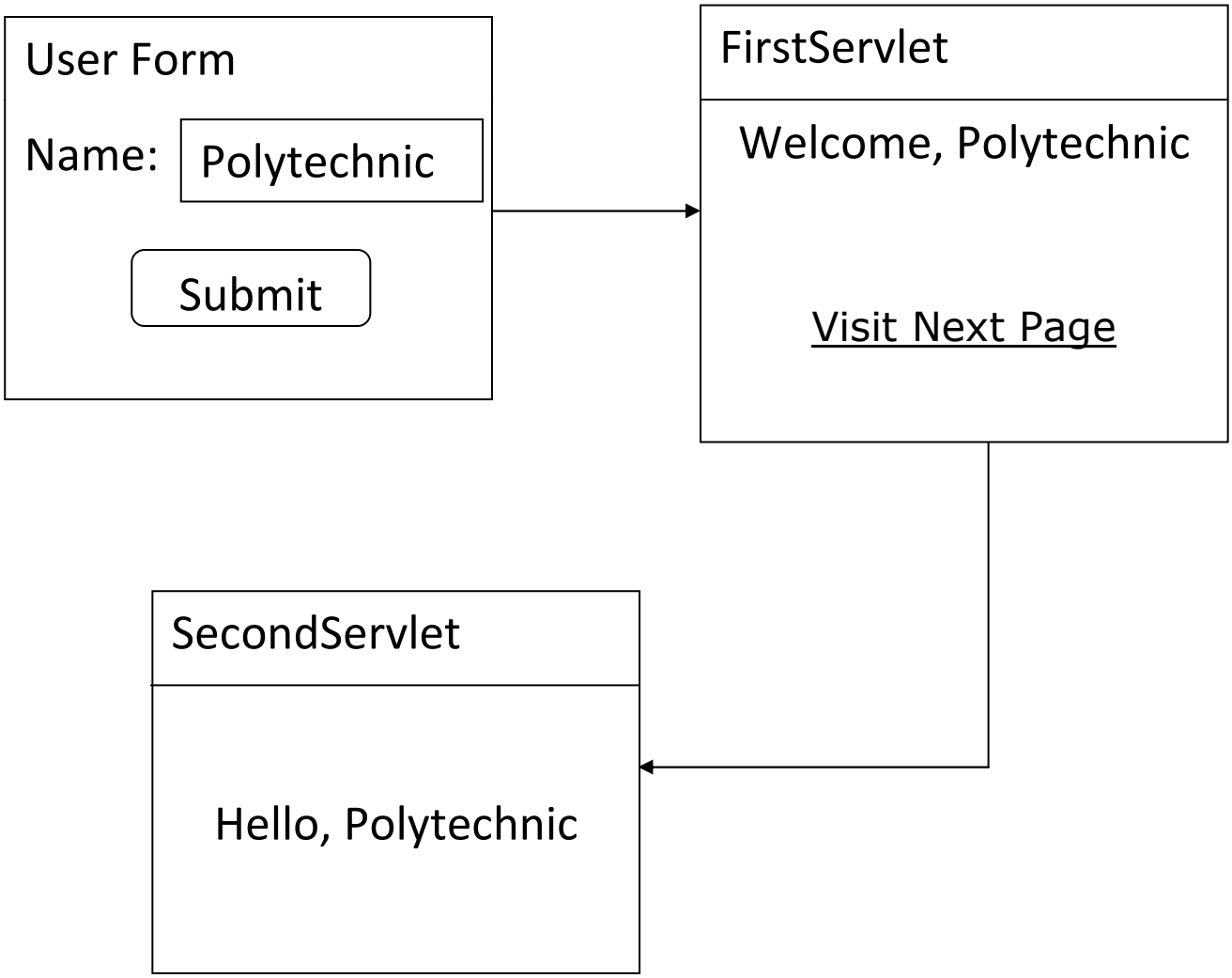
Advantage of URL Rewriting

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

Disadvantage of URL Rewriting

1- It will work only with links.
2- It can send Only textual information.

Example



Code:- UserForm.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

Code:- FirstServlet.java

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

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request,
                                HttpServletResponse response)
{
      try
      {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            String uName =request.getParameter("userName");
            out.print("Welcome "+ uName);
            out.print("<a href='servlet2? userName="+ uName +" '>");
            out.print("Visit Next Page </a>");

           out.close();
      }
      catch(Exception e)
      {
            System.out.println(e);
      }
   }
}
```

Code:- SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request,
                                HttpServletResponse response)
      try
      {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            //getting value from the query string
            String userName=request.getParameter("userName");
            out.print("Hello "+ userName);
            out.close();
      }
      catch(Exception e)
      {
            System.out.println(e);
      }
  }
}
```

HttpSession interface

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1- Bind objects
2- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

public HttpSession getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.

public HttpSession getSession(boolean create):Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1- **public String getId()**:Returns a string containing the unique identifier value.

2- **public long getCreationTime()**:Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

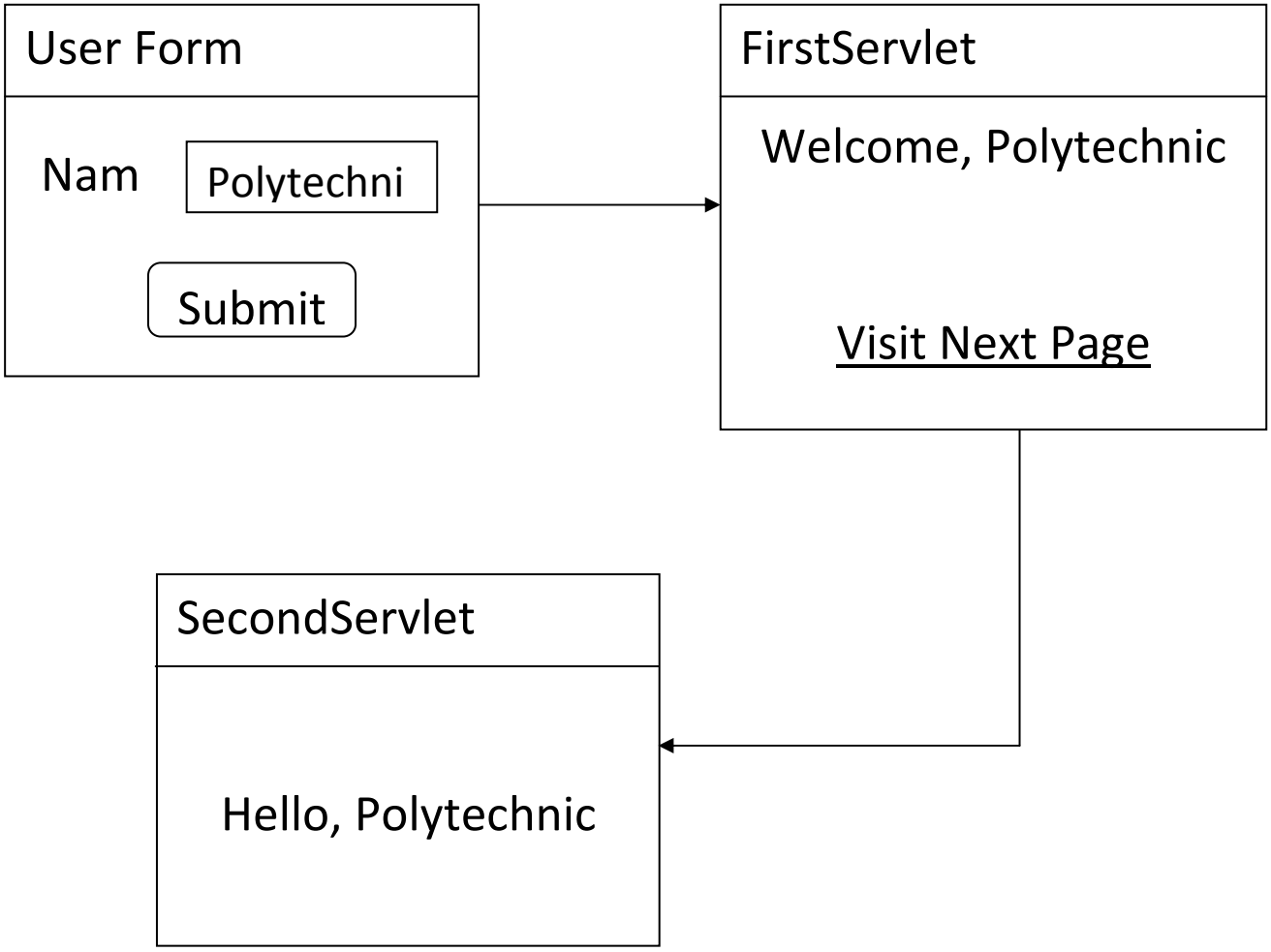
3- **public long getLastAccessedTime()**:Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

4- **public void invalidate()**:Invalidates this session then unbinds any objects bound to it.

Example

Code: UserForm.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```



Code: FirstServlet.java

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

public class FirstServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request,
                                    HttpServletResponse response){
        try
        {

        response.setContentType("text/html");
```

```
            PrintWriter out = response.getWriter();

        String userName =request.getParameter("userName");
        out.print("Welcome " + userName);

        HttpSession session=request.getSession();
        session.setAttribute("uname", userName);

        out.print("<a href='servlet2'>Visit Next Page </a>");

        out.close();

         }catch(Exception e){System.out.println(e);}
   }
}
```

Code: SecondServlet.java

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

public class SecondServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                                        HttpServletResponse response)
   try{

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

        HttpSession session=request.getSession(false);
        String username = (String)session.getAttribute("uname");
        out.print("Hello "+ username);

        out.close();

     }catch(Exception e){System.out.println(e);}
  }
}
```