Module 3

Java Servlet

Introduction to Servlet Architecture

- Servlet architecture comes under a java programming language used to create dynamic web applications.
- Mainly servlets are used to develop server-side applications.
- Servlets are very robust and scalable.
- Before introducing servlets, CGI (common gateway interface) was used.
- Servlets are used to perform client request and response tasks dynamically.

Introduction to Servlet Architecture

Servlets can be used to perform tasks like,

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

Introduction to Servlet Architecture

There are two types of Servlets-

1.Generic Servlets

2.HTTPServlets.

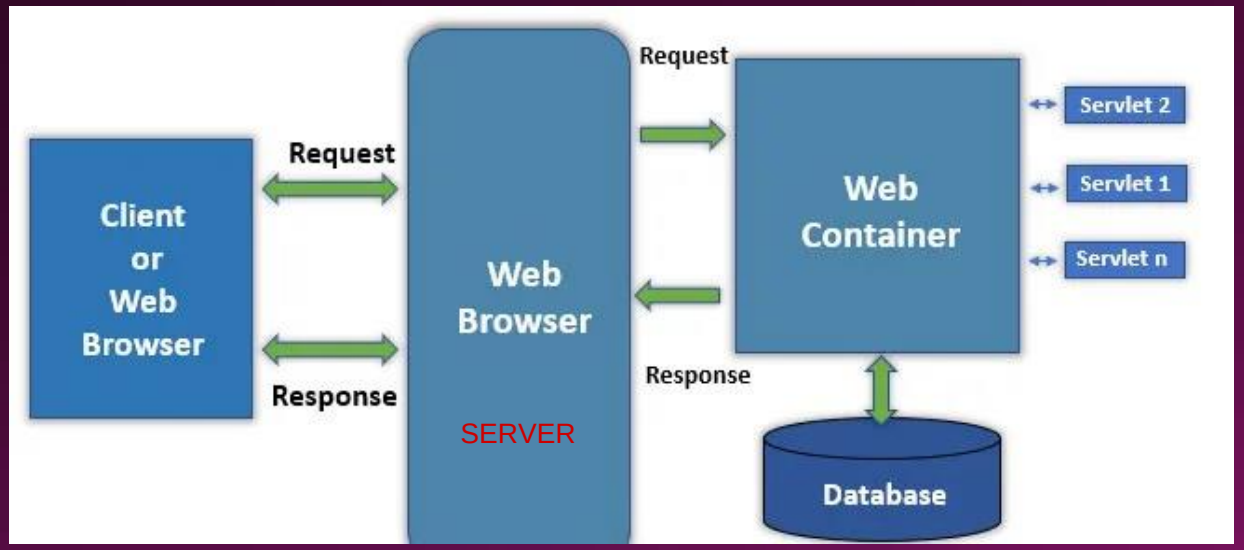
Servlets can be created in three possible ways

1. Implementing Servlet Interface
2. Extending Generic Servlet.
3. Extending HTTPServlet.

Three life cycle methods available with servlets are init(), service() and destroy().

Every servlet should override these methods.

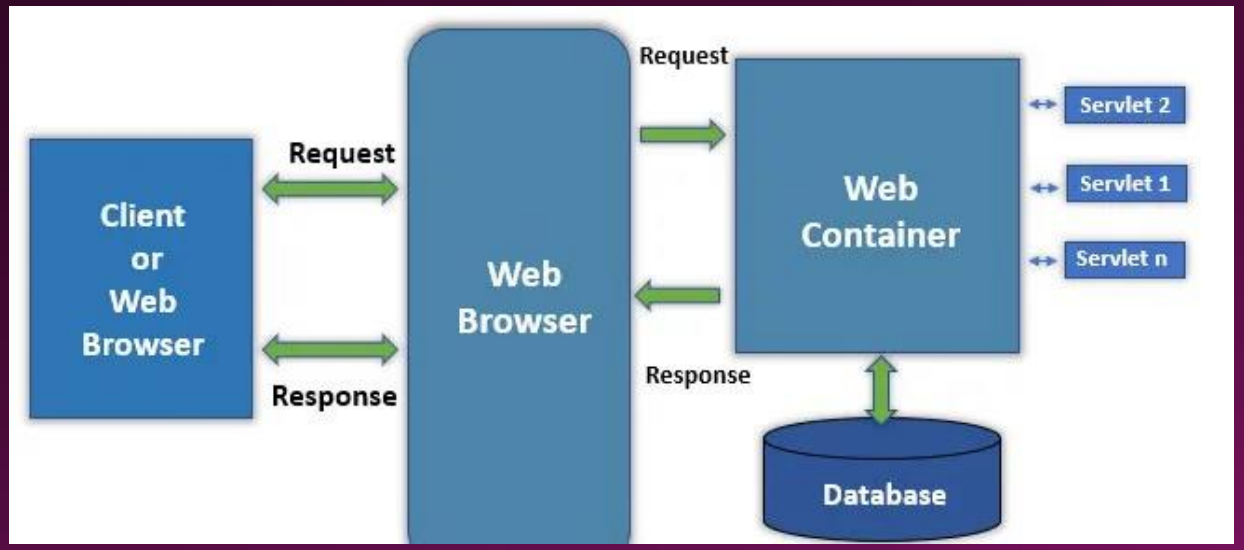
Components of Servlet Architecture



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

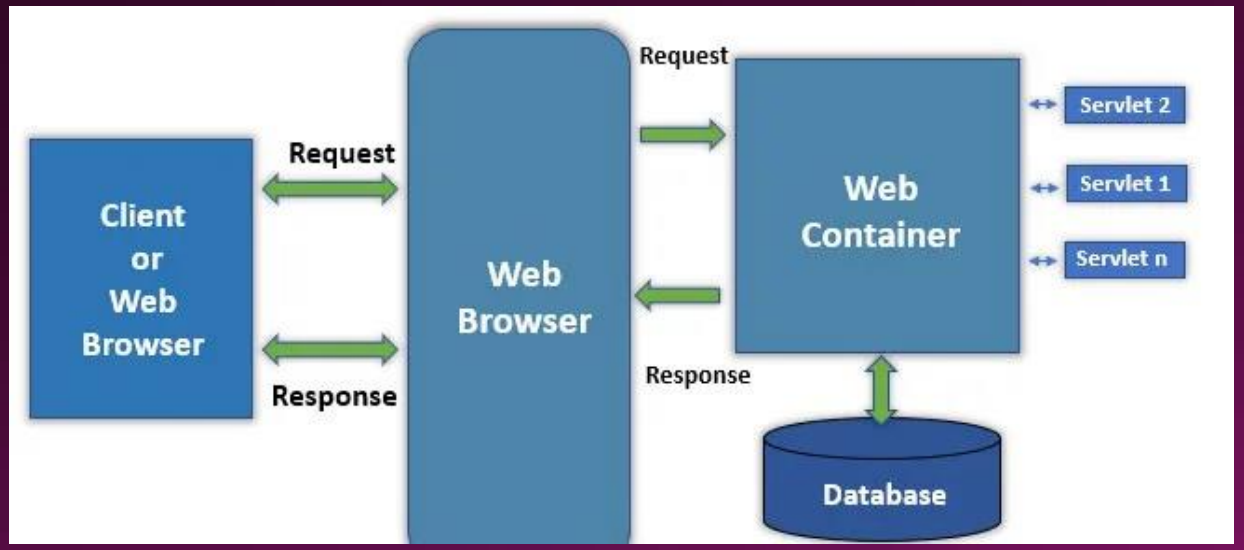
Components of Servlet Architecture



2. Web Server

- Web server controls how web user access hosted files, and it's responsible for processing user request and responses.
- Here server is software it understands URLs and HTTP protocol.
- Whenever a browser needs to host a file on the webserver, process client request using an HTTP request; if it finds the requested file sends it back to the browser through HTTP Response.
- There are two types' web servers Static and Dynamic webservers.
- in a static web server, it sends the file as it is, but in a dynamic web, the server-hosted file is updated before it is sent to the browser.

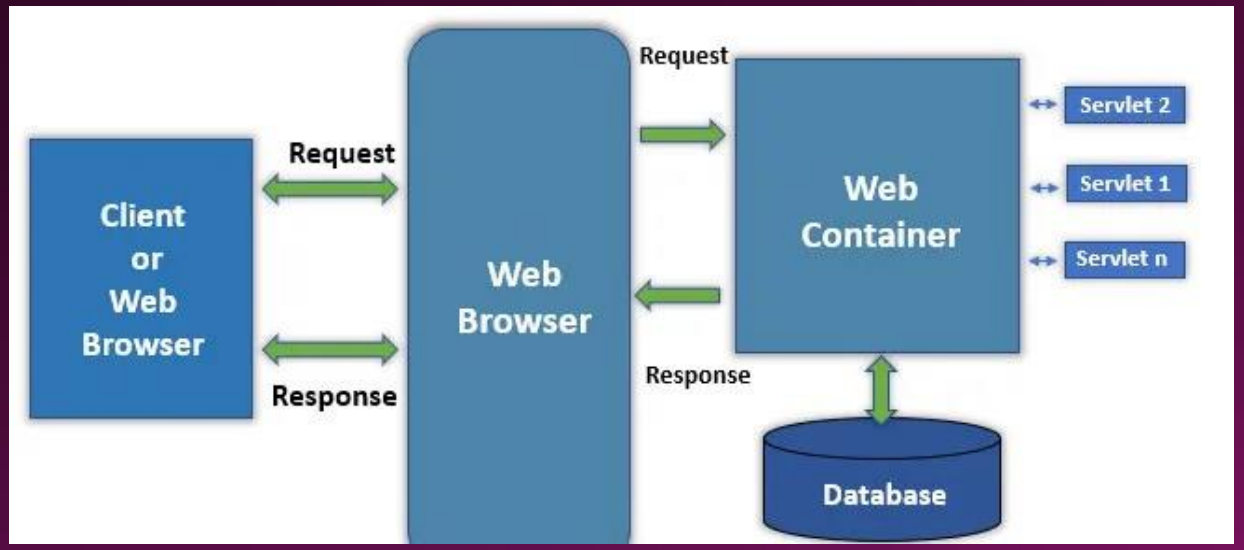
Components of Servlet Architecture



3. Web Container

- A web container is a component in the webserver it interacts with Java servlets.
- A web container is responsible for managing the lifecycle of servlets, and it also performs the URL mapping task.
- Web container handles the requests of servlets, JSP and other files at the server-side.
    - The important tasks performed by servlets are loading and unloading servlets,
    - creating and managing requests and response objects
    - performing servlet management's overall task.

Components of Servlet Architecture



Servlet Request Flow

1. The client sends a request.
2. Web Server accepts the request and forwards it to the web container.
3. Web container searches web.xml file for request URL pattern and gets the address of the servlet.
4. If the servlet is not yet instantiated, it will be instantiated and initialized by calling the init() method.
5. The container calls public service() by passing ServletRequest and ServletResponse objects.
6. Public service() method typecast ServletRequest and ServletResponse object to HttpServletRequest and HttpServletResponse objects respectively.
7. Public service() method calls protected service().
8. Protected service() method checks the client request & corresponding do__() method is called.
9. The request is handled by sending the result generated by do__() to the client.

Introduction to Servlet Architecture

Advantages:

- Servlets are server independent, as they are compatible with any web server.
- Compared to other server-side web technologies like ASP and JavaScript, these are server-specific.
- Servlets are protocol-independent, i.e. it supports FTP, SMTP, etc. Mainly it provides extended support to HTTP protocol functionality.
- Servlets are persistent because it remains in memory until explicitly destroyed this helps in several request processing and one database connection can handle several database requests.
- Servlets are portable; since the servlets are written in java, they are portable and supports any web server.
- Faster in execution, servlets are compiled into byte code execute more quickly compared to other scripting languages. Byte code conversion gives better performance and helps in type checking and error.

Uses of Servlet Architecture

1. Servlets are used to form data manipulation like accepting form data and generating dynamic HTML pages.
2. Servlets helps in developing server load balancing applications where load balancing is among different servers.
3. Servlets are used as the middle tier in enterprise network platforms for connecting the SQL database.
4. Servlets can be integrated with applets to provide high-level interactivity and dynamic web content generation.
5. Servlet is used to develop applications where they act as active agents in the middle tier, where they share data.
6. Since the servlet supports various protocols like HTTP, FTF, etc., this helps develop applications like file server applications and chat enabled applications.

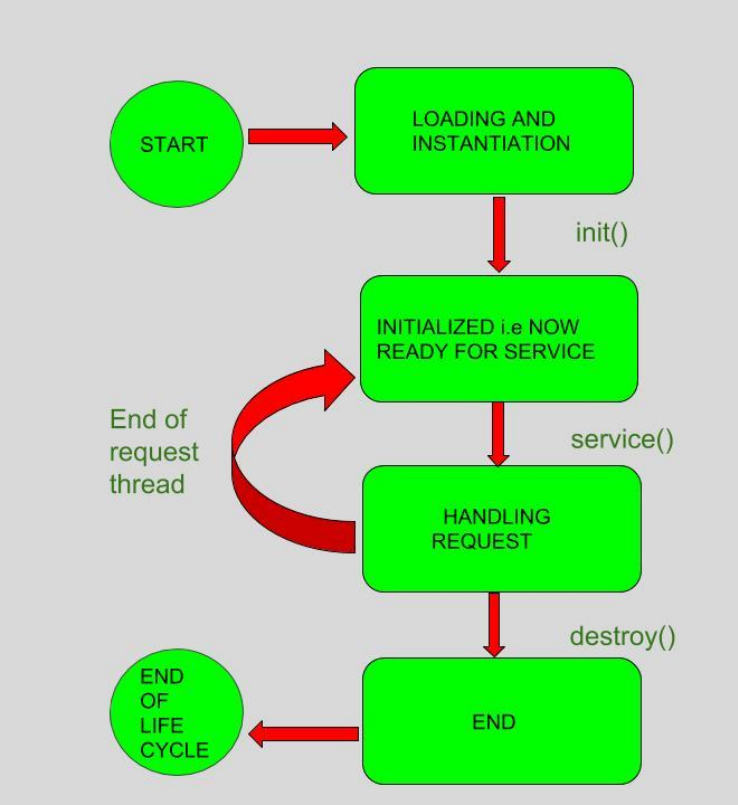
Life Cycle of Servlet

Life Cycle of Servlet

The Servlet life cycle mainly goes through four stages:

1. Loading a Servlet.
2. Initializing the Servlet.
3. Request handling.
4. Destroying the Servlet

Life Cycle of Servlet



Servlet Life Cycle:

Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.

The servlet life cycle consists these stages:

Servlet is borned

Servlet is initialized

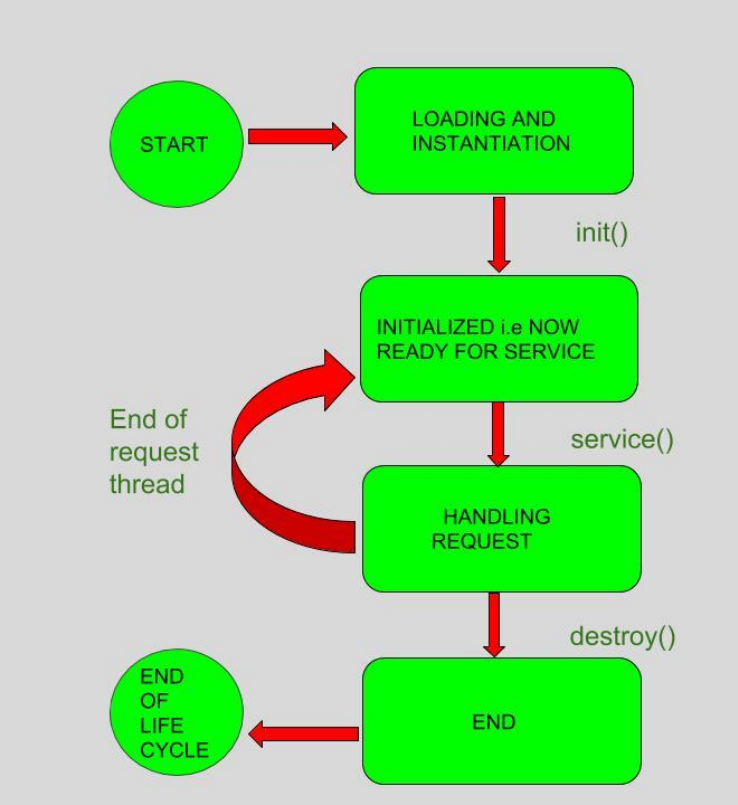
Servlet is ready to service

Servlet is servicing

Servlet is not ready to service

Servlet is destroyed

Life Cycle of Servlet



1. **Loading a Servlet:**

The first stage of the Servlet lifecycle involves loading and initializing the Servlet by the Servlet container.

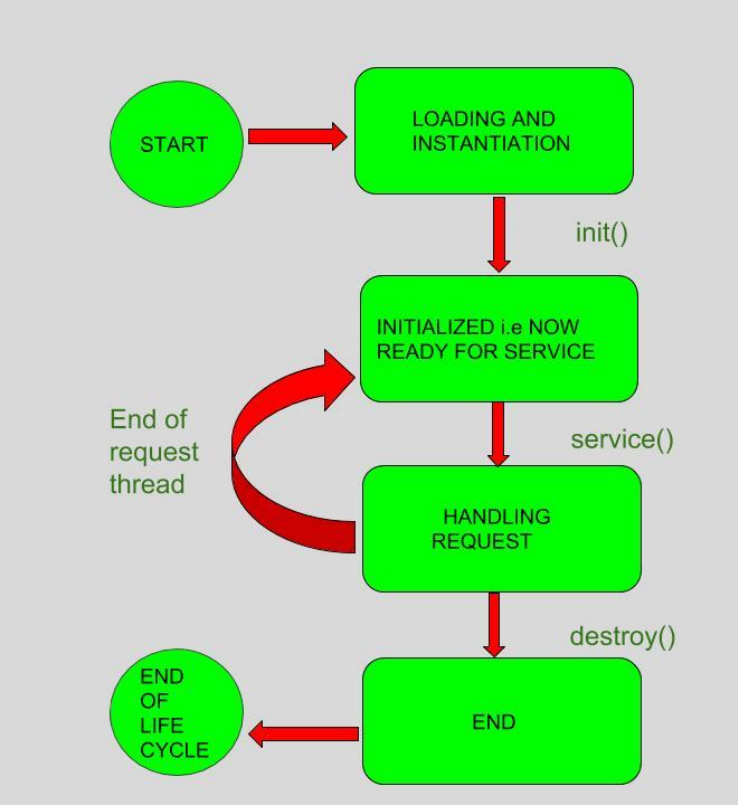
The Web container or Servlet Container can load the Servlet at either of the following two stages :

- Initializing the context, on configuring the Servlet with a zero or positive integer value.
- If the Servlet is not preceding stage, it may delay the loading process until the Web container determines that this Servlet is needed to service a request.

The Servlet container performs two operations in this stage :

1. Loading : Loads the Servlet class.
2. Instantiation : Creates an instance of the Servlet. To create a new instance of the Servlet, the container uses the no-argument constructor.

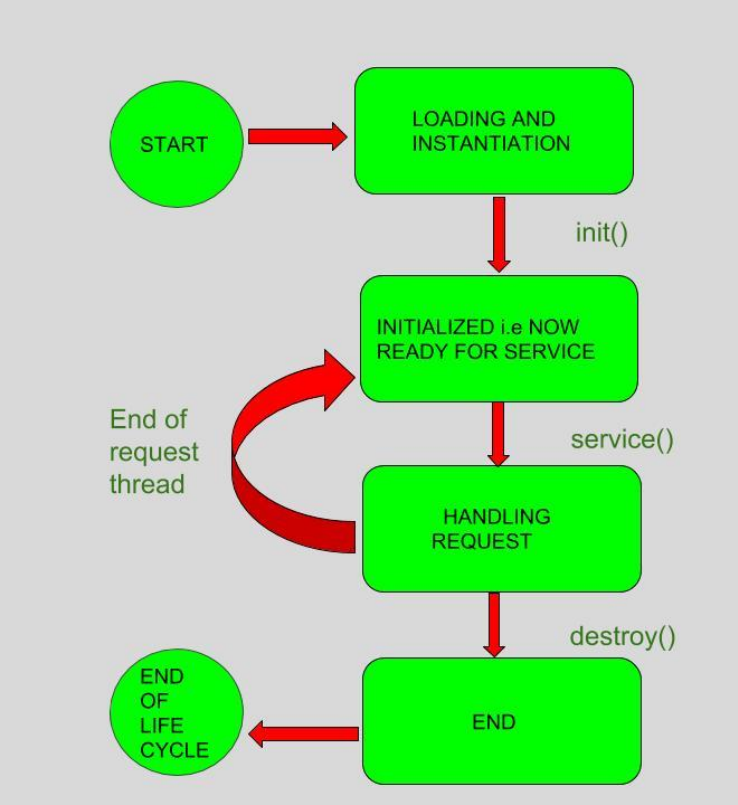
Life Cycle of Servlet



2. Initializing a Servlet:

- After the Servlet is instantiated successfully, the Servlet container initializes the instantiated Servlet object.
- The container initializes the Servlet object by invoking the Servlet.init(ServletConfig) method which accepts ServletConfig object reference as parameter.
- The Servlet container invokes the Servlet.init(ServletConfig) method only once, immediately after the Servlet.init(ServletConfig) object is instantiated successfully.
- This method is used to initialize the resources, such as JDBC datasource.
- Now, if the Servlet fails to initialize, then it informs the Servlet container by throwing the ServletException or UnavailableException..

Life Cycle of Servlet



3. Handling request:

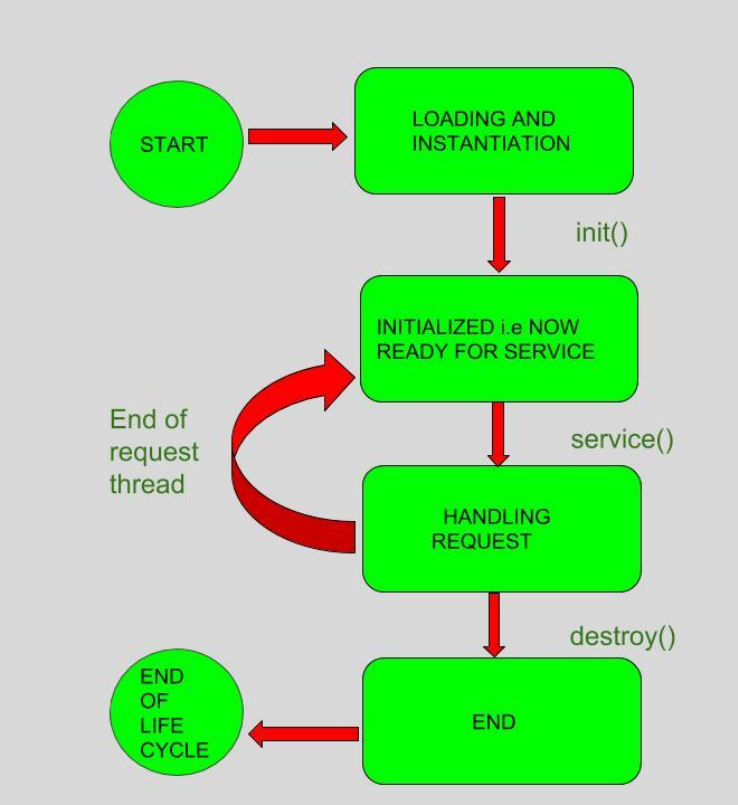
After initialization, the Servlet instance is ready to serve the client requests.

The Servlet container performs the following operations when the Servlet instance is located to service a request :

- It creates the ServletRequest and ServletResponse objects. In this case, if this is a HTTP request, then the Web container creates HttpServletRequest and HttpServletResponse objects which are subtypes of the ServletRequest and ServletResponse objects respectively.
- After creating the request and response objects it invokes the Servlet.service(ServletRequest, ServletResponse) method by passing the request and response objects.

The service() method while processing the request may throw the ServletException or UnavailableException or IOException...

Life Cycle of Servlet



4. Destroying a Servlet:

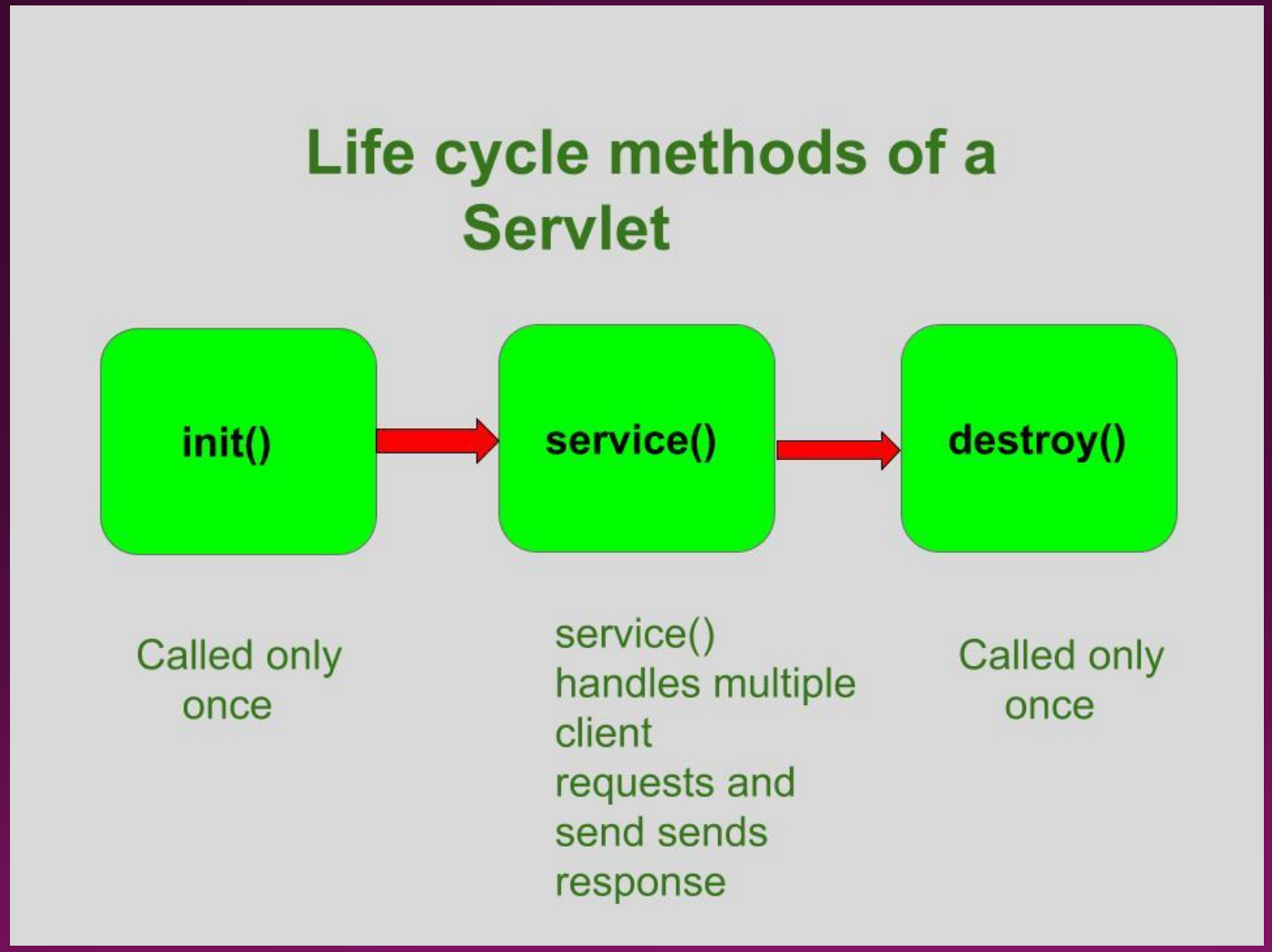
When a Servlet container decides to destroy the Servlet, it performs the following operations,

- It allows all the threads currently running in the service method of the Servlet instance to complete their jobs and get released.
- After currently running threads have completed their jobs, the Servlet container calls the destroy() method on the Servlet instance.

After the destroy() method is executed, the Servlet container releases all the references of this Servlet instance so that it becomes eligible for garbage collection.

Servlet Life Cycle Methods

Servlet Life Cycle Methods



Servlet Life Cycle Methods

```
//init() method

public class MyServlet implements Servlet
{
        public void init(ServletConfig config) throws ServletException
        {
                //initialization code
        }
         //rest of code
}
```

1. init() method:

- The **Servlet.init()** method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.
- A servlet's life begins here .
- This method is called only once to load the servlet.
- Since it is called only once in it's lifetime, therefore "connected architecture" code is written inside it because we only want once to get connected with the database.

Servlet Life Cycle Methods

```
// service() method

public class MyServlet implements Servlet
{
        public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException
        {
                // request handling code
        }
        // rest of code
}
```

2. service() method:

- The service() method of the Servlet is invoked to inform the Servlet about the client requests.
- This method uses ServletRequest object to collect the data requested by the client.
- This method uses ServletResponse object to generate the output content
- The service() method is the most important method to perform that provides the connection between client and server.
- The web server calls the service() method to handle requests coming from the client (web browsers) and to send response back to the client.
- This method determines the type of Http request (GET, POST, PUT, DELETE, etc.) .
- This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete(), etc. as required.
- This method accepts two parameters.
    - req is the ServletRequest object which encapsulates the connection from client to server
    - resp is the ServletResponse object which encapsulates the connection from server back to the client

Servlet Life Cycle Methods

```
// destroy() method
public void destroy()
{
        // Finalization code...
}
```

3. destroy() method:

- The destroy() method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.
- As soon as the destroy() method is activated, the Servlet container releases the Servlet instance.
- The destroy() method is called only once.
- It is called at the end of the life cycle of the servlet.
- This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.
- When this method is called, the garbage collector comes into action.

Creation of Servlet Application

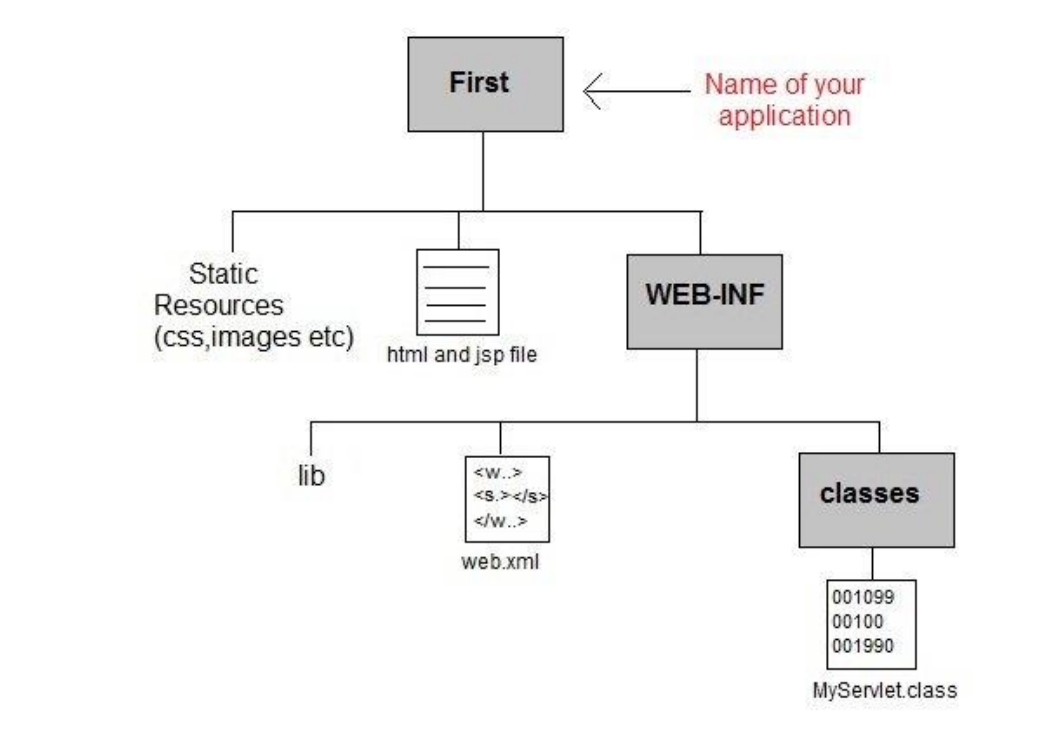
Steps

Install any Web Server – Let say Apache Tomcat Server

After installing Tomcat Server on your machine follow the below mentioned steps :

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployment Descriptor for your application
5. Start the server and deploy the application

1. Creating Directory Structure



- Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.
- Create the directory structure shown in figure 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 (deployement descriptor) file is kept under WEB-INF folder.

2. Creating Servlet

- There are three different ways to create a servlet.
  - By implementing Servlet interface
  - By extending GenericServlet class
  - By extending HttpServlet class
- But mostly a servlet is created by extending HttpServlet abstract class.
- HttpServlet gives the definition of service() method of the Servlet interface.
- The servlet class that we will create should not override service() method.
- Our servlet class will override only doGet() or doPost() method.
- 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.

NOTE: By default a request is Get request.

2. Creating Servlet

```
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 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:

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

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

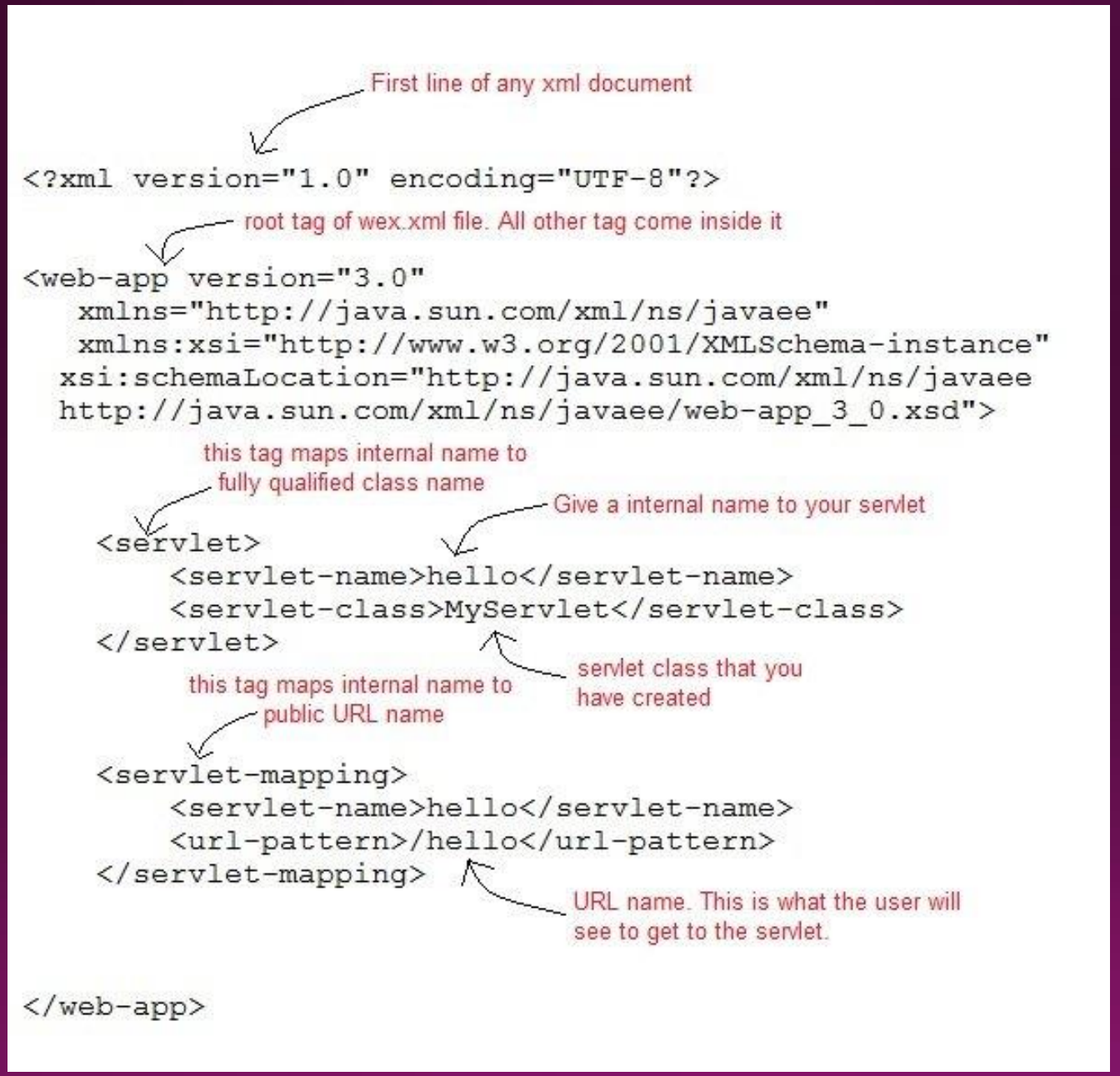
4. Create Deployment Descriptor

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.

4. Create Deployment Descriptor



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.

1. Right Click on My Computer, go to Properites.
2. Go to Advanced Tab and Click on Enviroment Variables... button.
3. Click on New button, and enter JAVA_HOME inside Variable name text field and path of JDK inside Variable value text field.
4. Click OK to save.

7. Run Servlet Application

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

Reading Form Data using Servlet

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.

GET Method Example using URL

```
// 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 public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";
```

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

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Given Here is the HelloForm.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 –

GET Method Example using URL

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

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

http://localhost:8080/HelloForm?first_name = ZARA&last_name = ALI

Given Here is the HelloForm.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 –

GET Method Example using URL

```
<servlet>
  <servlet-name>HelloForm</servlet-name>
  <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloForm</servlet-name>
  <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

1. Compile file HelloForm.java

$ javac HelloForm.java

If everything goes fine, above compilation would produce HelloForm.class file.

2. Copy class file in

<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes

3. Create entries in web.xml file

File is located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

4. Test the working with URL

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

GET Method Example using URL

```
<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>
```

5. Create HTML Form

Create form which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

6. Save the file as Hello.htm

7. Put file in the location

<Tomcat-installationdirectory>/webapps/ROOT

8. Access the HTML form as

http://localhost:8080/Hello.htm

POST Method Example using URL - Servlet will handle both GET and POST method

```
// 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 public \"-//w3c//dtd html 4.0 " +
      "transitional//en\">\n";
```

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

POST Method Example using URL – Form using POST method

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

- Post method does not append parameters to action URL: /servlet/MyServlet
- Instead, parameters are sent in body of request where the password is not visible as in GET method
- POST requests are not idempotent
- From Mathematics – an idempotent unary operator definition: whenever it is applied twice to any element, it gives the same result as if it were applied once.
- Cannot bookmark them
- Are not safely repeatable
- Can’t be reloaded
- browsers treat them specially, ask user

Cookies Handling

Cookies

Cookies are text files stored on the client computer and they are kept for various information tracking purpose.

Java Servlets transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

1. Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
2. Browser stores this information on local machine for future use.
3. 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.

The Anatomy of Cookie

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

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
  path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

- The Set-Cookie header contains a name value pair, a GMT date, a path and a domain.
- The name and value will be URL encoded.
- The expires field is an instruction to the browser to "forget" the cookie after the given time and date.
- If the browser is configured to store cookies, it will then keep this information until the expiry date.
- If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.

Setting Cookies with Servlet

A servlet will then have access to the cookie through the request method request.getCookies() which returns an array of Cookie objects.

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);
```

Session Tracking

Session Handling

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server:

1. Cookies
2. Hidden Form Fields
3. URL Rewriting
4. The HttpSession Object

Cookies

- A webserver can assign a unique session ID as a cookie to each web client
- For subsequent requests from the client they can be recognized using the recieved cookie.
- This may not be an effective way because many time browser does not support a cookie,
- so It would not be recommended to use this procedure to maintain the sessions.

Hidden Form Fields

- A web server can send a hidden HTML form field along with a unique session ID as follows –

  <input type = "hidden" name = "sessionid" value = "12345">

- This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data.
- Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.
- This could be an effective way of keeping track of the session
- but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting

- You can append some extra data on the end of each URL that identifies the session,
- The server can associate that session identifier with data it has stored about that session.

  For example, with http://tutorialspoint.com/file.htm;sessionid = 12345,

  The session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.
- URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies.
- The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

The HttpSession Object

- servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
- The servlet container uses this interface to create a session between an HTTP client and an HTTP server.
- The session persists for a specified time period, across more than one connection or page request from the user.
- You would get HttpSession object by calling the public method getSession() of HttpServletRequest, as below –

```
HttpSession session = request.getSession();
```

- You need to call request.getSession() before you send any document content to the client

Content

- Client, Server and Communication
- The Internet
- Basic Internet Protocols
- World Wide Web