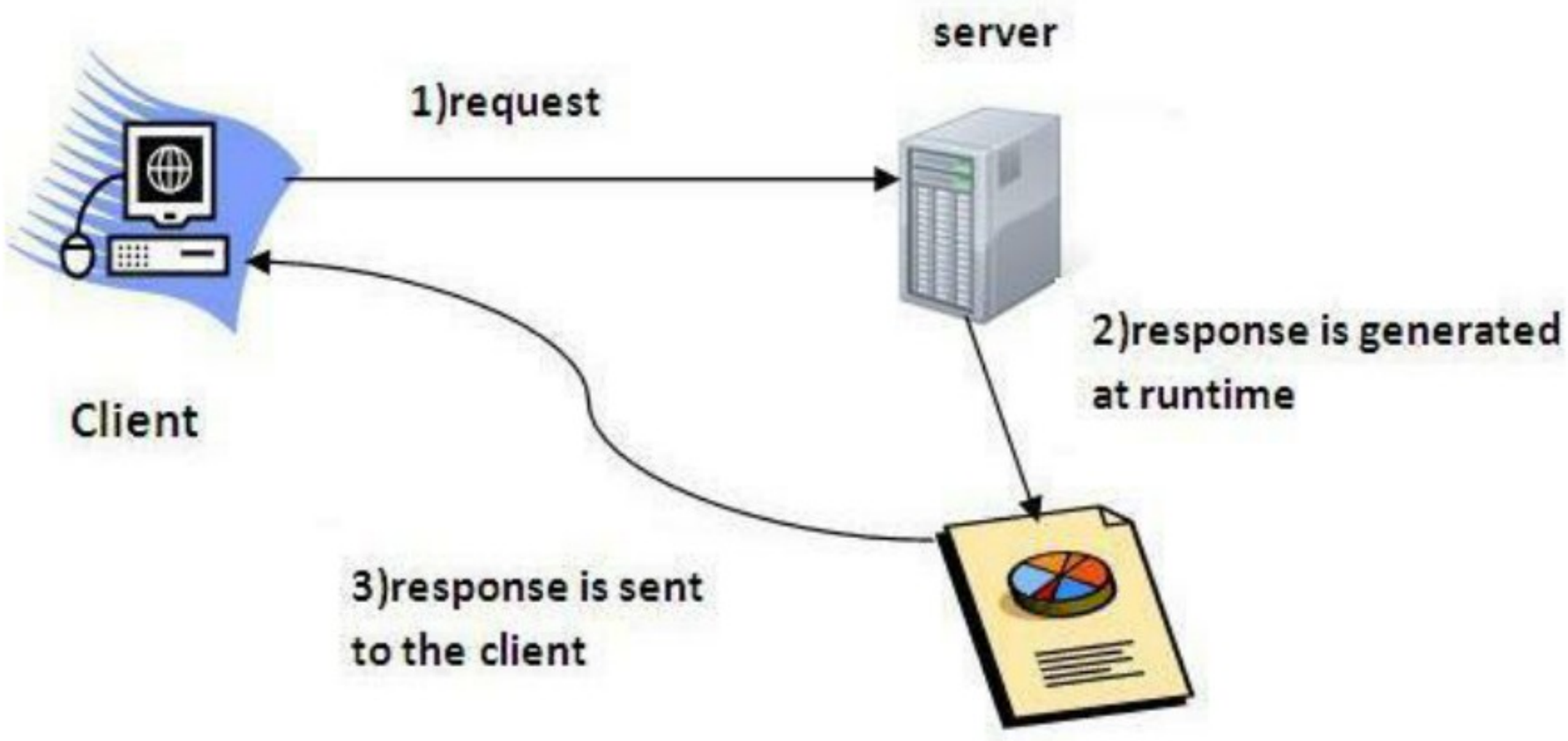
Servlet

- Servlet technology is used to create web application, resides at server side and generates dynamic web page.
- Before Servlet, CGI (Common Gateway Interface) was popular as a server-side programming language.
- But there were many disadvantages such as creating separate process for each request, platform dependent code (C, C++), high memory usage and slow performance.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming request.
- Servlets purely works on the basis of **Request- Response** protocol.
- HTTP Requests are sent from browser to server and server responds to the client browser as HTTP Response.
- Servlet is a web component that is deployed on the server to create dynamic web page.
- As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.



Advantage of Servlet

- The web container creates threads for handling the multiple requests to the servlet.
- Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low.
- The basic benefits of servlet are as follows:
  1. **Better Performance:** It creates a thread for each request not process.
  2. **Portability:** It uses java language.
  3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
  4. **Secure:** It uses java language.

Life Cycle of Servlet

- The web container maintains the life cycle of a servlet instance. Its life cycle have following phases:
  1) Servlet class is loaded.
  2) Servlet instance is created.
  3) init method is invoked.
  4) service method is invoked.
  5) destroy method is invoked.

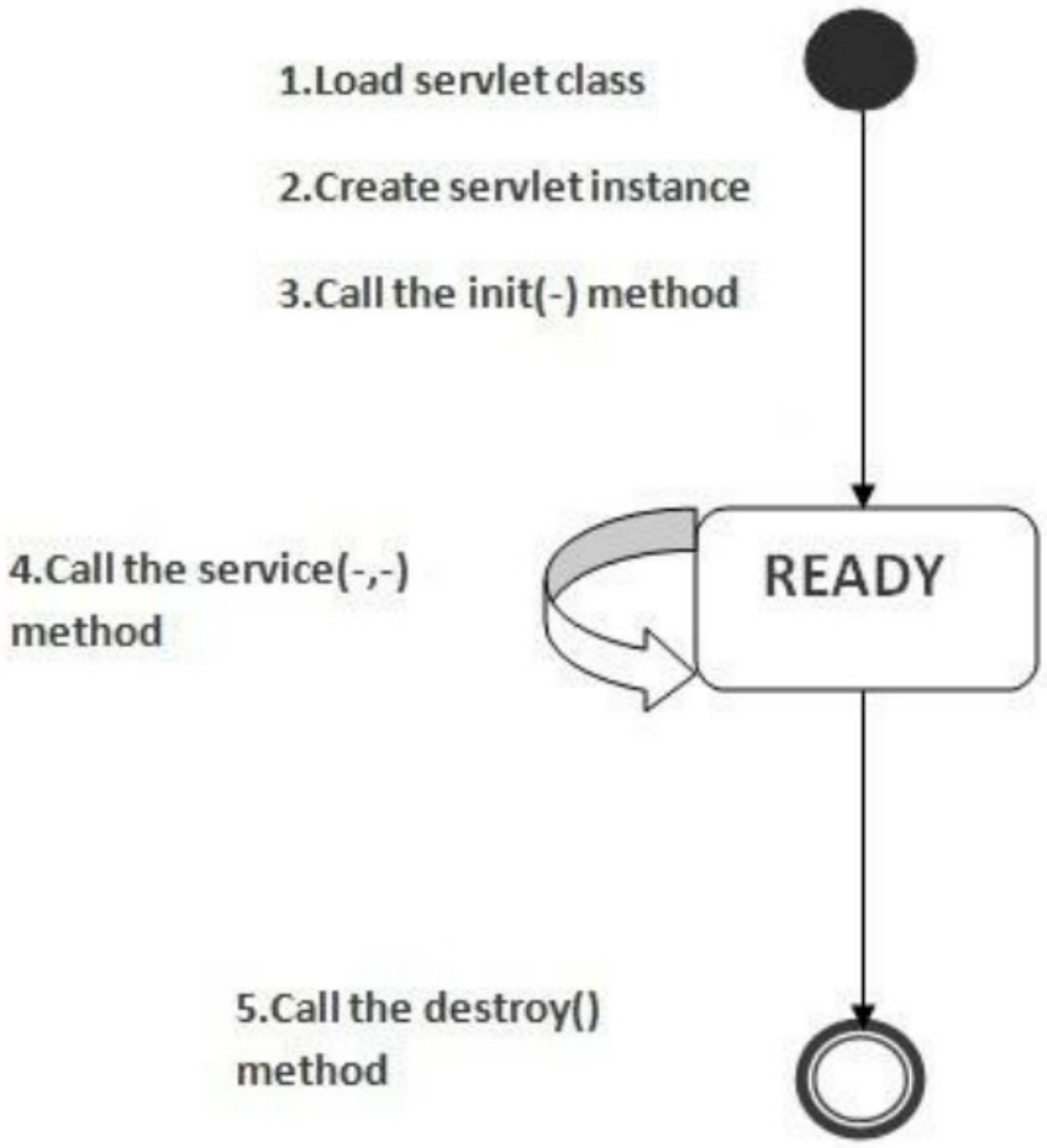


Fig: Life Cycle of Servlet

1) **Loading Servlet Class**
   - The servlet class is loaded when the first request for the servlet is received by the web container.
2) **Servlet instance creation**
   - The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

- The web container calls the init() method only once after creating the servlet instance. The init method is used to initialize the servlet.

public void **init** (**ServletConfig** config) **throws** ServletException

4) service method is invoked

- The web container calls the service () method each time when request for the servlet is received.
- If servlet is not initialized, it follows the first three steps as described above then calls the service method.
- If servlet is initialized, it calls the service method. The servlet is initialized only once.
- The service () method will then call the doGet() or doPost() methods based on the type of the HTTP request (Get or Post).

public void **service**(**ServletRequest** request, **ServletResponse** response)
throws ServletException, IOException

5) destroy method is invoked

- The web container calls the destroy method before removing the servlet instance from the service.
- It clean up any resource for example memory, thread etc.

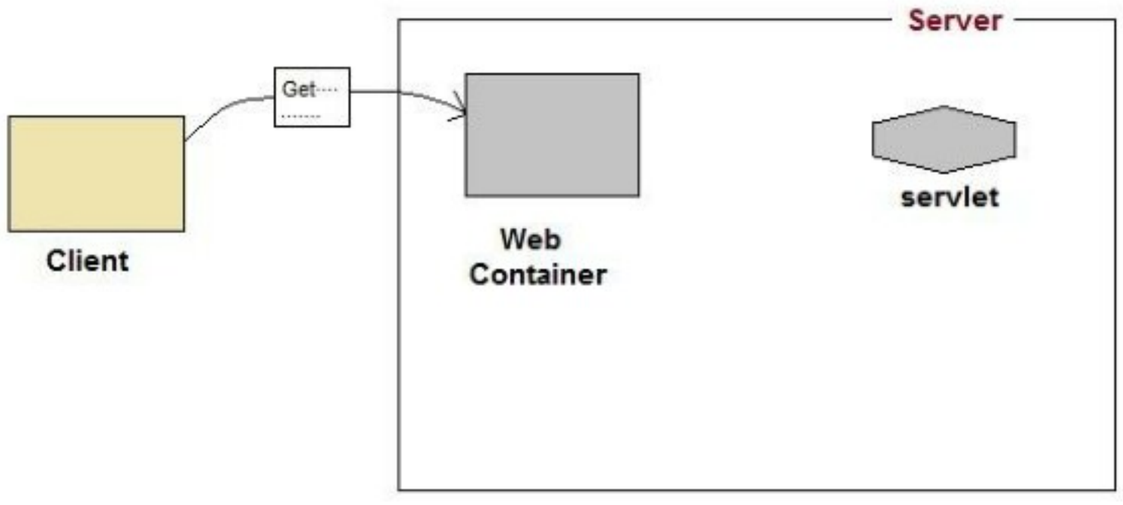
public void **destroy** ()

How a Servlet Application Works

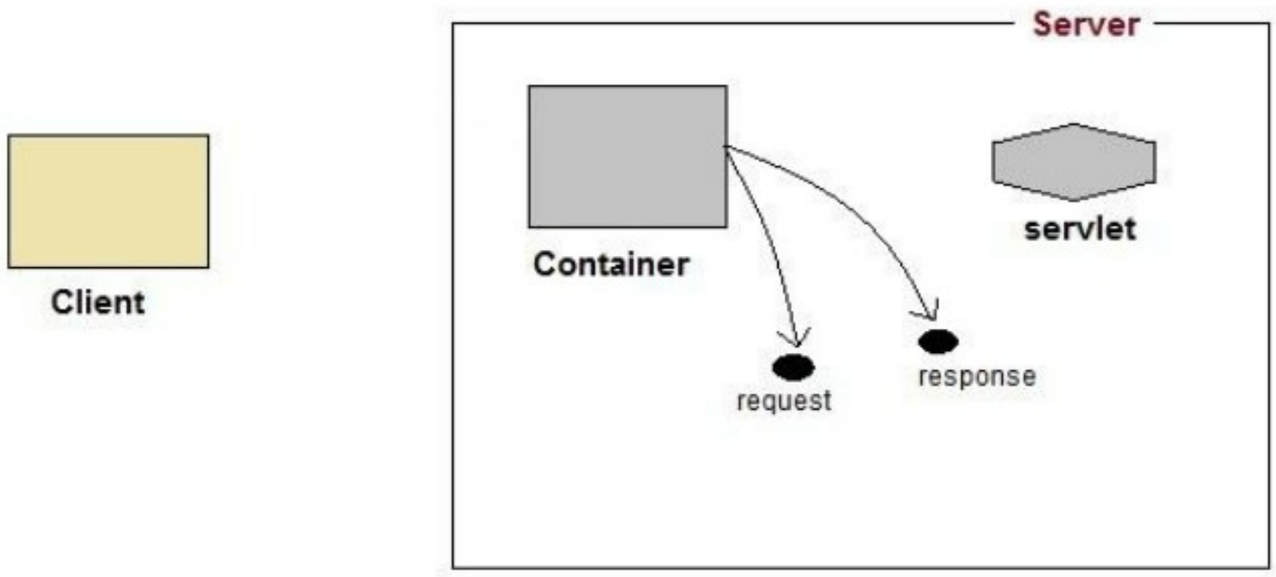
- **Web container** is responsible for managing execution of servlets and JSP pages for Java EE application.
- When a request comes in for a servlet, the server hands the request to the Web Container.
- Web Container is responsible for instantiating the servlet or creating a new thread to handle the request.
- Web Container is responsible for to get the request and response to the servlet.
- The container creates multiple threads to process multiple requests to a single servlet.
- **Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

❖ **Client request propagate through following steps and get the response**:

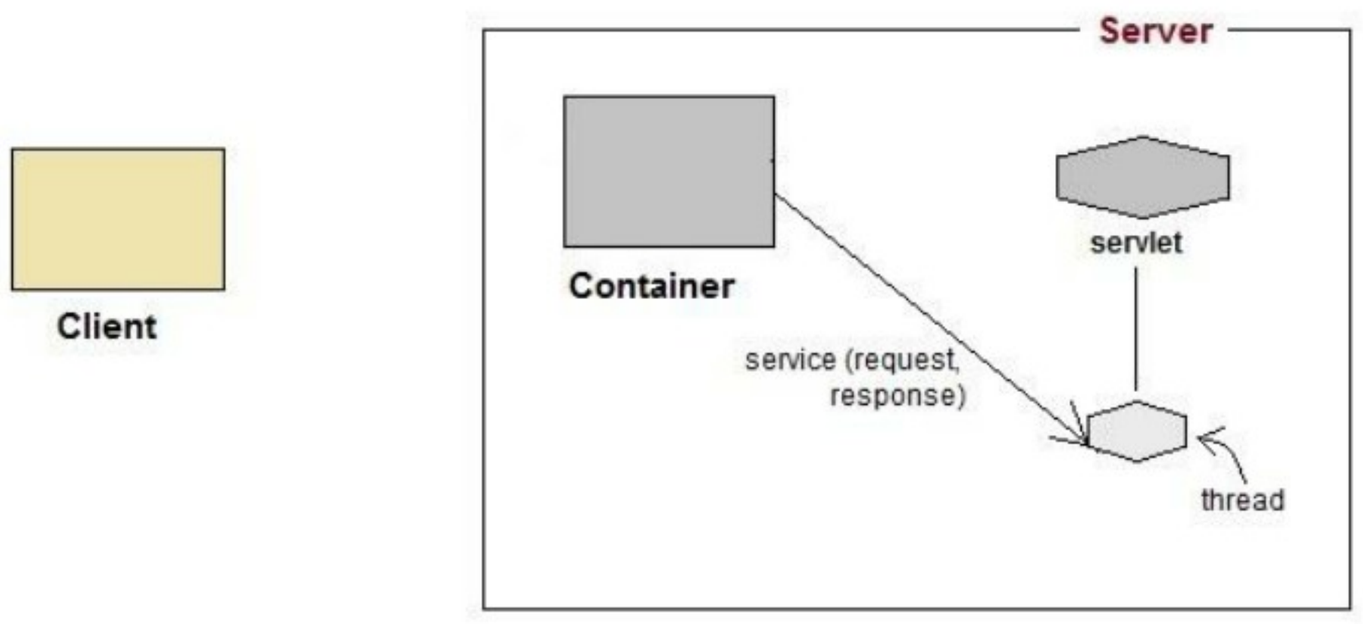
1) User sends request for a servlet by clicking a link that has URL to a servlet.



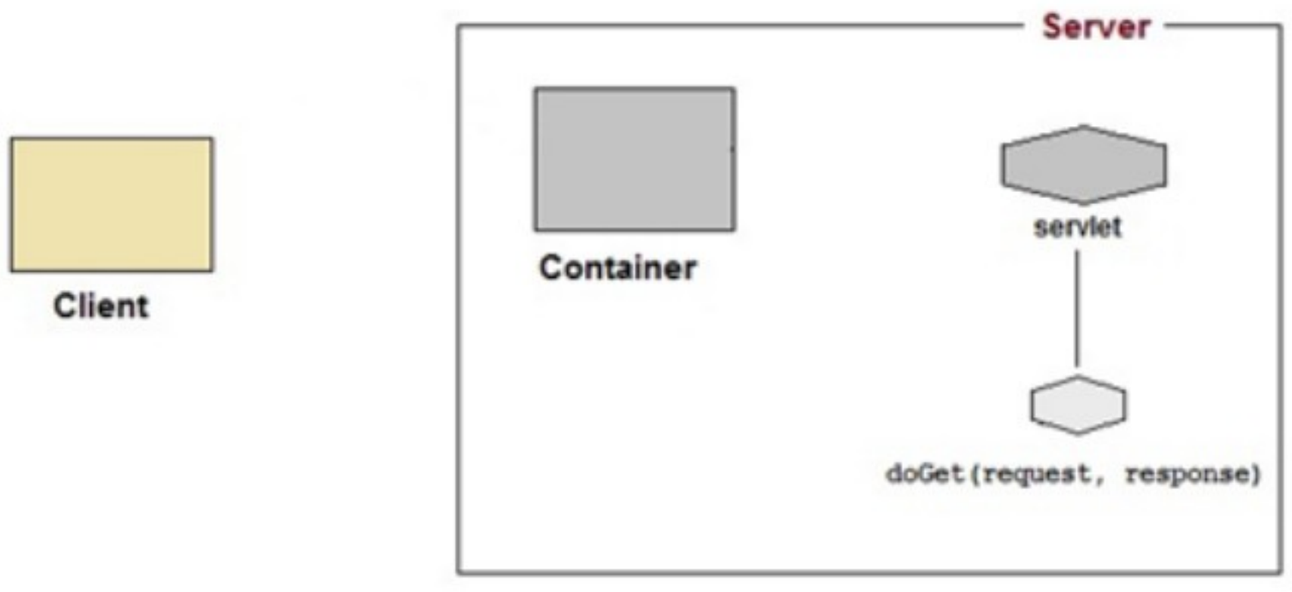
2) The container finds the servlet using deployment descriptor and creates two objects :
   a) **HttpServletRequest**
   b) **HttpServletResponse**



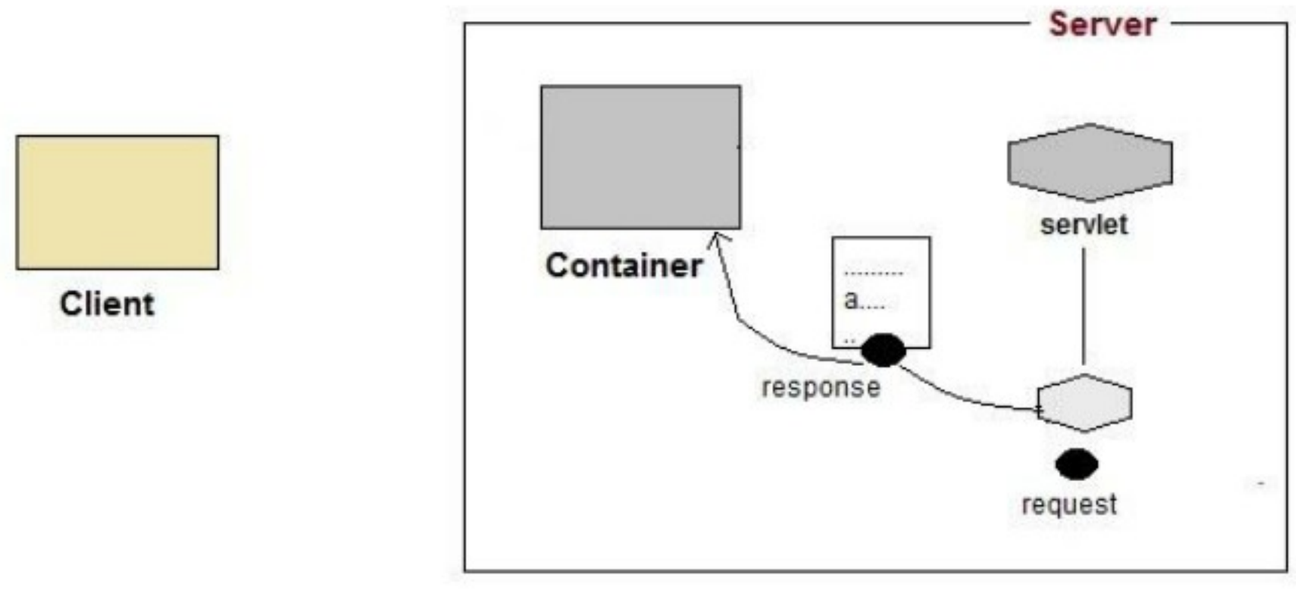
3) Then the container creates or allocates a thread for that request and calls the Servlet's `service()` method and passes the **request, response** objects as arguments.



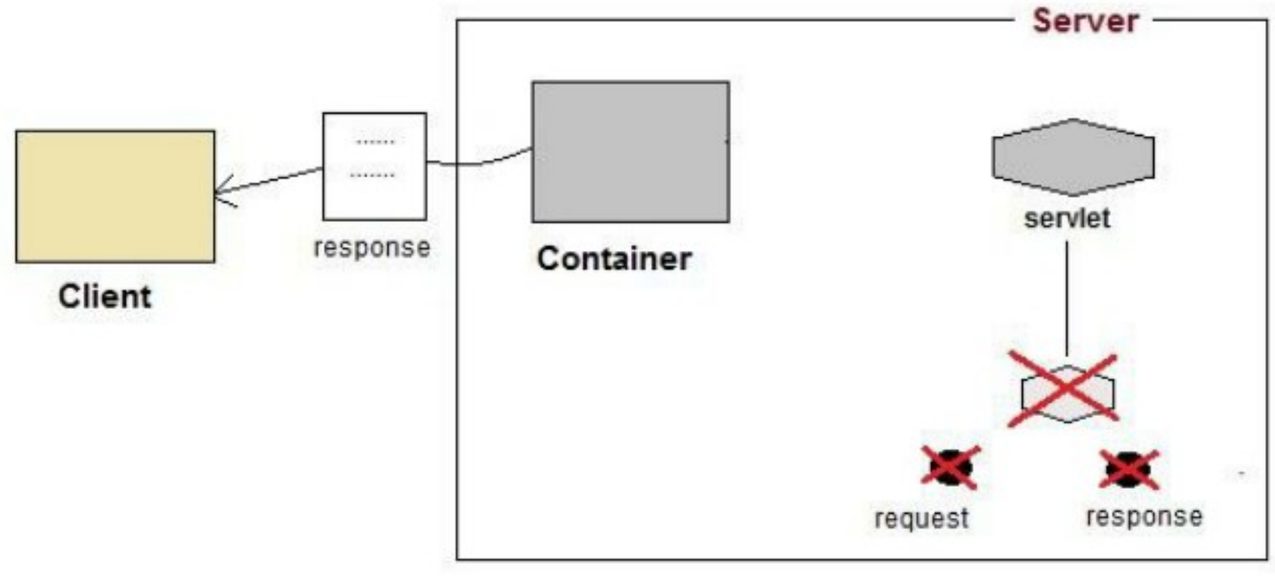
4) The `service()` method, then decides which servlet method, `doGet()` or `doPost()` to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the `service()` will call Servlet's `doGet()` method.



5) Then the `Servlet` uses response object to write the response back to the client.



6) After the `service()` method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



The Java Servlet Development Kit

- JSDK (Java Servlet Development Kit) is a package containing all the classes and interfaces needed to develop servlets.
- JSDK also contains a web server and servlet engine to test your creations. The servlet engine provided in JSDK is a basic one and free.
- JSDK is a Java Servlet Runner program that runs on your workstation, and allows you to test servlets you have written without running a web server.
- A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them.

Steps to install JSDK

- JSDK download from the official website.

- Run the file you downloaded and install on **C:\JSDK2.1**. This contains the source code for the servlet classes, a simple Web server to use as a test environment for servlets, full documentation and examples, and assorted utilities.
- Now, you need to set the **CLASSPATH** environment variable. For that type following command in command prompt :

  set CLASSPATH=%CLASSPATH%;c:\jsdk2.1\servlet.jar

- To start JSDK server write following command in command prompt under **C:\JSDK2.1** directory :

  Startserver

- To stop JSDK server write following command in command prompt under **C:\JSDK2.1** directory :

  Stopserver

- You can change the JSDK server property in the **default.cfg** file .
- You can specify the directory where the JSDK server looks for servlets by editing the file default.cfg located in **C:\JSDK2.1** directory.

  Example : To set this directory to the examples directory that comes with the JSDK, change the **server.docbase** property to examples in `default.cfg file`:

  server.docbase = examples

- Also we can change port number when the JSDK server clashes with another server already installed on the system by changing port number in default.cfg file.
- The servlets directory is located at :

  docbase/WEB.INF/servlets

- After the JSDK server is started, paste below given URL to browser and get desire output.

  http://localhost:8080/servlet/ HelloWorldExample

Apache Tomcat

- It is an open source web server and servlet container developed by the Apache Software Foundation (ASF).
- It implements the Java Servlet and the JavaServer Pages (JSP) specifications and provides a pure Java HTTP web server environment for java code to run.
- To install and configure Apache Tomcat on the local system follow below given steps:
  - Download Apache Tomcat from official website.
  - Create Folder and unzip apache-tomcat-8.0.32.

    Like, **F:\TomcatProject\apache-tomcat-8.0.32**

- Create a Environment Variable, new under system variable named **JAVA_HOME** and value set as **c:\Program Files\Java\jdk1.8.0_{xx}**
- To Configure Tomcat Server update below given file under **/conf** folder::

  server.xml ---->> port no to 9999 (If you want to change port number o/w optional)

  web.xml ---->> 'listings' from "false" to "true" for the "default servlet"

  context.xml ---->> Replace <Context> to <context reloadable="true">

  tomcat-users.xml ---->> Add below line to <tomcat-users> tag

  <role rolename="tomcat"/>

  <user username="tomcat" password="tomcat" roles="tomcat"/>

- **To Start Tomcat Server :**
  - Open cmd and propogate through the path upto bin directory inside apache-tomcat-8.0.32 folder which we previously unzipped and execute below command:

    startup.bat

- Congratulation Tomcat Server Has Been Started.
- **To access Tomcat Server :**
  - Open browser and Paste given URL : **http://localhost:9999**
- **To Stop Tomcat Server :**

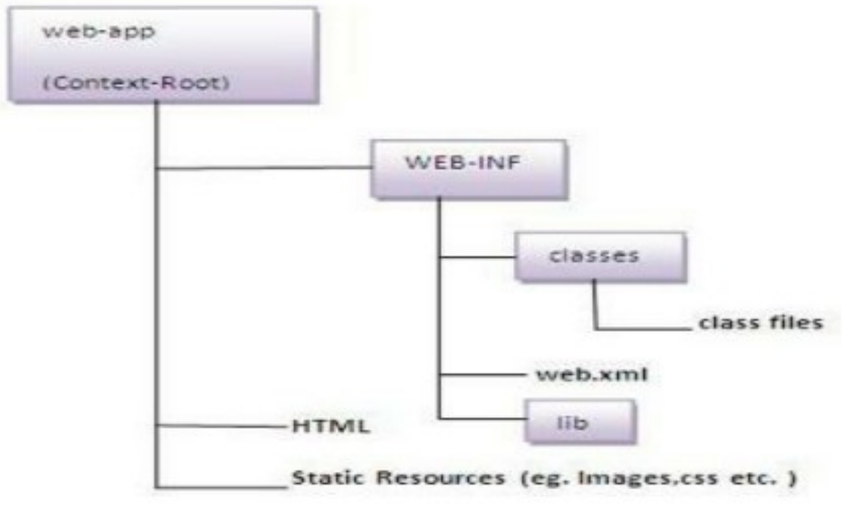
  Shutdown.bat

❖ Write a program of simple Servlet. Create and compile servlet source code, start a web browser and request the servlet, and deployment in 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 Deployement Descriptor for your application
  5) Start the server and deploy the application
  6) Access the servlet

1) Create the directory structure

- All file must be saved under **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 a Servlet

- There are three different ways to create a servlet.
  - By implementing **Servlet** interface
  - By extending **GenericServlet** class
  - By extending **HttpServlet** class
- The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost etc.
- By default a request is Get request.

Solution:

```
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 World !!!</h1>");
                out.println("</body></html>");
        }
}
```

- Write above code in a notepad file and save it as **MyServlet.java** anywhere on your system. Compile it and paste the class file into **WEB-INF/classes/** directory which is inside **Tomcat/webapps** directory.

3) Compiling a Servlet

- To compile a Servlet a JAR file is required. Different servers require different JAR files.

Server	Jar file
Apache Tomcat	servlet-api.jar
Glassfish	javaee.jar
JBoss	javaee.jar

- Steps to compile a Servlet
  - Download **servlet-api.jar** file.
  - Paste the **servlet-api.jar** file inside **Java\jdk\jre\lib\ext** directory.
  - Compile the Servlet class.

    Example: F:\> javac MyServlet.java

  - After compiling the java file, paste the class file of servlet in **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.
- Some elements of web.xml files are as follows:

Elements	Description
<web-app>	Represents the whole application.
<servlet>	It is sub element of <web-app> and represents the servlet.
<servlet-name>	It is sub element of <servlet> represents the name of the servlet.
<servlet-class>	It is sub element of <servlet> represents the class of the servlet.
<servlet-mapping>	It is sub element of <web-app>. It is used to map the servlet.
<url-pattern>	It is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

- Now we will see how to create a simple **web.xml** file for our web application & paste it in WEB-INF folder.

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

</web-app>
```

First line of any xml document

root tag of wex.xml file. All other tag come inside it

this tag maps internal name to fully qualified class name

Give a internal name to your servlet

servlet class that you have created

this tag maps internal name to public URL name

URL name. This is what the user will see to get to the servlet.

5) Start the server and Deploy the application

- Double click on the **startup.bat** file under apache-tomcat-x.x.x\bin\ directory Or execute the following command on your windows machine using Command prompt to start tomcat server.

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

- Starting Tomcat Server for the first time
  - Firstly you need to set **JAVA_HOME** in the Environment variable. The following steps will show you how to set it :
    - Right Click on **My Computer**, go to **Properties**.
    - Go to **Advanced** Tab and Click on **Environment Variables** button.
    - Click on **New** button, and enter **JAVA_HOME** in Variable name and **path of JDK** in Variable value. Click OK to save.

6) Run Servlet Application

- Open Browser and write following pattern of URL:

http://hostname:portno/contextroot/urlpatternofservlet

Example : **http:localhost:8080/First/hello**

❖ Demonstrate the example that read database and display table record using servlet by using javax.servlet Package.

- To access the database and its table using servlet, firstly we will create table and insert record in that table using command prompt.
- **Steps to create table Employees in Student database and insert records into it.**

Step 1: Open a Command Prompt and change to the installation directory as follows:

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

Step 2: Login to database as follows:

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ****
mysql>
```

Step 3: Create the table Employee in TEST database as follows:

```
mysql> use Student;
mysql> create table Student_Detail
        (
           Id int not null,
          Name varchar (255),
          Branch varchar (255)
        );
```

Query OK, 0 rows affected (0.08 sec)

Step 4: Create Data Records in Employees table

```
mysql> INSERT INTO Student_Detail VALUES (1, 'Emily', 'Computer');
```

Query OK, 1 row affected (0.05 sec)

```
mysql> INSERT INTO Student_Detail VALUES (2, 'Joseph', 'Computer');
```

Query OK, 1 row affected (0.00 sec)

- Below example shows how to read database and display records of table using servlet.

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

```
public class DemoServletDatabase extends HttpServlet
{
        public void doGet (HttpServletRequest req, HttpServletResponse res) throws
        IOException, ServletException
        {
                String url = "jdbc:mysql://localhost:3306/Student";
                String query = "select * from Student_Detail";

                //Create an object of outputstream Printwriter()
                PrintWriter out  =  res.getWriter();

                //Set response content type
                res.setContentType("text/html");
                out.println ("<html><body>");
                try
                {
                        out.println("Trying to connect"+</br>);
                        Class.forName("com.mysql.jdbc.Driver");
                        Connection con  =  DriverManager.getConnection(url, "root",
                                           "root");
                        out.println("Connection successful"+</br>);
                        Statement  stmt  =  con.createStatement();
                        ResultSet  rs  =  stmt.executeQuery (query);
                        out.println("<table border=1 >");
                        out.println("<tr><th>Id</th><th>Name</th><th>Branch</th><tr>");
                        while (rs.next())
                        {
                                int id  = rs.getInt("Id");
                                String Name = rs.getString("Name");
                                String Branch = rs.getString("Branch");

                                //Display values in tabular format
                                out.println("<tr><td>" + id + "</td><td>" + Name +
                                "</td><td>" + Branch + "</td></tr>");

                }
                out.println("</table>");
                out.println("</html></body>");
```

```
                con.close();
            }
            catch (Exception e)
            {
                out.println("Error...");
            }
        }
    }
```

- To compile **DemoServletDatabase.java** use following command:

  F:\ > javac **DemoServletDatabase.java**

- Store the class file **DemoServletDatabase.class** into **/WEB-INF/classes** folder**.**
- **Copy below code in web.xml :**

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

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

- **Now call this servlet using URL from browser:**

  http://localhost:8080/demoservletdb/DemoServletDatabase

Output:

Trying to connect
Connection successful

Id	Name	Branch
2	Emily	Computer
3	Joseph	Computer

❖ **Difference between GET/doGet and POST/doPost method:**

GET	Post
GET is less secure compared to POST because data sent is part of the URL. So it's saved in browser history.	POST is a little safer than GET because the parameters are not stored in browser history.
GET method should not be used when sending passwords or other sensitive information.	POST method used when sending passwords or other sensitive information.
GET method is visible to everyone and has limits on the amount of information to send.	POST method variables are not displayed in the URL and have no limit on the amount of information to send.
Only ASCII characters allowed.	The POST method can be used to send ASCII as well as binary data.
Parameters remain in browser history because they are part of the URL.	Parameters are not saved in browser history.

❖ **Difference between Applet and Servlet method:**

Applet	Servlet
It is part of Core Java.	It is part of Advance java.
Applets are the Java Programs which run in web browser, on the Client machine.	Servlets are the Java Programs which run on the web or application Server.
An Applet can use the user interface classes like AWT or Swing.	Servlet does not have a user interface.
Applets are treated as untrused and they have a limited permission to run in the client browser.	Servlet can be treated as Server-side applets. It runs in a servlet container which is deployed in the webserver.
An applet is downloaded in to the client's machine and run on the client's browser.	Servlet runs on the server and transfers the results back to the client when it is done
Using applets, the entire code of the applet has to be transferred to the client. Therefore it consumes more network bandwidth than servlet.	While Servlets, which transfers only the results to the client.
The packages used for Applets are: **java.applet**, **java.awt**	The packages used for Servlets are : **javax.servlet, javax.servlet.http**

Session Tracking

- A session refers to the entire interaction between a **client** and a **server** from the time of the **client's first request**, which generally begins the session, to the time the session is terminated.
- **Session** simply means a particular interval of time.
- We all know that **HTTP** is a **stateless** protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests.
- Since, HTTP and Web Server both are stateless, the only way to maintain a session is when some **unique information** about the session (session id) is passed between server and client in every request and response.
- **Session Management** is a mechanism used by the **Web container** to store session information for a particular user.
- There are four different techniques used by servlet application for session management.
  1) **HttpSession**
  2) **Cookies**
  3) **Hidden form field**
  4) **URL Rewriting**

HttpSession

- **HttpSession** object is used to store entire session with a specific client.
- **Container** creates a **session id** for each user. The container uses this unique 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.
- The **HttpServletRequest** interface provides following methods to get the object of **HttpSession**:

Method	Description
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)	Returns the current session associated with this request or, if there is no current session and create is true, returns a new session.
public String **getRequestedSessionId**()	Gets the ID assigned by the server to the session.

- Important methods of the **HttpSession** as given below:

Method	Description
long **getCreationTime**()	Returns the time when the session was created, measured in milliseconds.
String **getId**()	Returns a string containing the unique identifier assigned to the session.
long **getLastAccessedTime**()	Returns the last time the client sent a request associated with the session.
void **invalidate**()	Used to destroy the session.
boolean **isNew**()	Returns true if the session is new else false.
void **setMaxInactiveInterval** (int interval)	Specifies the time, in seconds, **after** that servlet container will **invalidate** the session.
int **getMaxInactiveInterval**()	Returns the **maximum time interval**, in **seconds**. (The servlet container will keep this session open between client accesses.)
Object **getAttribute**(String name)	Returns the **object bound** with the specified **name** in this session, or **null** if no object is bound under the name.
void **setAttribute**(String name, Object value)	Binds an object to this session, using the name specified.
void **removeAttribute**(String name)	Removes the object bound with the specified name from this session.

Cookie

- **Cookies** are small pieces of information that are sent in **response** from the **web server to the client**.
- **Cookies** are the simplest technique used for **storing client state**.
- By default, each request is considered as a new request.
- In this 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.
- **javax.servlet.http.Cookie** class provides the functionality of cookies.
- There are 2 types of cookies in servlets:
  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.

❖ **Advantages:**

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

❖ **Disadvantages:**

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

- Constructor of the Cookie class as given below:

Constructor	Description
Cookie()	Constructs a cookie.
Cookie(String name, String value)	Constructs a cookie with a specified **name** and **value**.

- Method of the Cookie class as given below:

Method	Description
void **setMaxAge**(int expiry)	This method sets how much time (in seconds) should elapse before the **cookie expires**.
String **getName**()	Returns the name of the cookie. The name cannot be changed after creation.
String **getValue**()	Returns the **value** of the cookie.
void **setName**(String name)	Changes the **name** of the cookie.
void **setValue**(String value)	Changes the **value** of the cookie.
void **addCookie**(Cookie ck)	Method of **HttpServletResponse** interface is used to add cookie in response object.
Cookie[] **getCookies**()	Method of **HttpServletRequest** interface is used to return all the cookies from the browser.

Hidden form field

- Hidden Form Field is **a hidden (invisible) textfield** is used for maintaining the state of an user.
- We store the information in the hidden field and get it from another servlet.
- This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Syntax: <input type="hidden" name="uname" value="James">

URL Rewriting

- If the client has disabled cookies in the browser then session management using cookie won't work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.
- In URL rewriting, a token(parameter) is added at the end of the URL.
- The token consist of **name/value pair** separated by an **equal**(=) sign.

Example : URL?uname=xyz&pass=def

- The **Web Container** will fetch the extra part of this requested URL and use it for session management.
- The getParameter() method is used to get the parameter value at the server side.

Difference between ServletConfig and ServletContext

ServletConfig	ServletContext
ServletConfig object is **one per servlet** class.	ServletContext object is **global** to entire **web application.**
Object of ServletConfig will be created during **initialization** process of the servlet.	Object of ServletContext will be created at the time of **web application deployment.**
ServletConfig object is **public** to a particular **servlet only** and destroy once execution of the servlet is completed.	ServletContext object will be available, and it will be **destroyed** once the application is **removed from the server**.
We need to request servlet explicitly, in order to create ServletConfig object for the **first time**.	ServletContext object will be available even **before giving the first request**.
<init-param> tag will be appear under ***<servlet-class>*** tag in **web.xml**.	***<context-param>*** tag will be appear under ***<web-app>*** tag in web.xml.
getServletConfig() method is used to get the config object	**getServletContext()** method is used to get the context object.
ServletConfig is an interface which has a set of methods like **getInitParameter(String name), getInitParameterNames(), getServletName() & getServletContext().**	ServletContext is an interface which has a set of methods like **getServletName**(**String name**), **getServletContext**(), **getInitParameter**(), **getInitParameterNames**().