Servlet

A Servlet is a Java program that runs on a web server and handles HTTP requests (like GET, POST) from a web browser.

It receives requests → processes them → sends response back.

One-line meaning:

Servlet = Java program for building web applications.

Example:

When a user clicks Login, the form data goes to a Servlet, which checks username/password and sends output to the browser.

Normal Java Program

Runs locally on your computer.

Executed using:

java MyProgram

Takes input from keyboard, file, console.

Produces output on console, file, or GUI.

Servlet

Runs inside a web server (Tomcat, Jetty, JBoss).

Automatically executed when a browser sends a request.

Input comes from HTTP Request (browser).

Output is HTML/JSON/XML sent back to browser.

How They Receive Input

Normal Java Program Input:

Scanner sc = new Scanner(System.in);

int x = sc.nextInt();

Input comes from keyboard.

Servlet Input:

Servlet receives HTTP request data:

Form data

URL parameters

Cookies

JSON body

Example:

String name = request.getParameter("username");

4 How They Give Output

Normal Java Program Output:

System.out.println("Hello");

Output appears on console.

Servlet Output:

response.getWriter().println("<h1>Hello User</h1>");

Output is sent to browser (Chrome, Firefox).

Who Calls Them?

Normal Java Program

You must run it manually:

java MyProgram

 You control execution.

Servlet

Servlet is never called manually.

It is always called by the server when a request comes.

Example:

http://localhost:8080/login

Browser calls servlet → Tomcat executes it → response goes back.

Servlet API

The Servlet API is a set of interfaces and classes provided by Java (Jakarta/Javax package) which allows you to:

- Create servlets
- Handle HTTP requests (GET, POST…)
- Read data sent by browser
- Send responses back
- Manage cookies & sessions
- Control servlet lifecycle
- Interact with server
- Forward/redirect pages

The 3 Most Important Packages in Servlet API

1.javax.servlet.*

Contains core interfaces like:

- Servlet
- ServletConfig
- ServletContext
- RequestDispatcher

Used for:

- Basic servlet lifecycle
- Configurations
- Application-level communication

2. javax.servlet.http.*

Contains HTTP-specific classes:

- HttpServlet
- HttpServletRequest

- HttpServletResponse
- HttpSession
- Cookie

Used for:

- Handling browser requests
- Sessions & cookies
- GET/POST processing

3. Servlet Container (Example: Tomcat)

Provides implementations for all these interfaces.

Meaning:

- You write the Servlet (class)
- Tomcat runs it using its Servlet API implementation

Key Interfaces & Classes

1. HttpServlet

The base class for all servlets.

You override:

- doGet()
- doPost()
- doPut()
- doDelete()

2. HttpServletRequest

Used to read data sent by browser:

- Form fields
- Query params
- Headers
- Cookies

Example:

String name = request.getParameter("username");

3. HttpServletResponse

Used to send output to browser (HTML, JSON).

Example:

response.getWriter().println("Login Success");

4.RequestDispatcher

RequestDispatcher is an object used to:

forward

- Transfer the request from one servlet/JSP to another servlet/JSP.

include

- Insert another page (header/footer/etc.) into the current response.

Servlet Lifecycle

A servlet goes through these phases:

- Load (container decides to load the servlet)
- Instantiate (container creates one servlet object)
- Initialize (init(ServletConfig)) — called once
- Serve requests (service() → doGet() / doPost() etc.) — called per request, many times, multi-threaded
- Destroy (destroy()) — called once before servlet is removed/unloaded
- Garbage collection (after destroy() the object becomes eligible for GC)

1. Load

- The Servlet container (Tomcat) loads the servlet class into JVM.
- When does it happen?
- On first request to that servlet (default lazy loading), OR
- At server startup if you configured load-on-startup (eager loading).
- How to configure eager load:
- @WebServlet(value="/login", loadOnStartup=1) or in web.xml with <load-on-startup>1</load-on-startup>.

2. Instantiate

- Container uses new to create one instance of your servlet class.
- There's typically one instance per servlet definition, not one instance per request.

3. Initialize — init(ServletConfig) / init()

- Called once immediately after instantiation.
- Purpose: perform one-time costly setup: open DB connections, read init params, allocate thread pools, read config files.
- You can access servlet init parameters via getServletConfig().getInitParameter("name") or getServletContext().

4. Service — service(HttpServletRequest, HttpServletResponse)

- Called for every request.
- HttpServlet.service() dispatches to doGet(), doPost(), etc., based on request method.
- Important: multiple threads call service() concurrently on the same servlet instance — so do not use mutable instance fields to store request-specific data.
- Use local variables inside doGet/doPost for per-request state.

5. Destroy — destroy()

- Called once by the container when it decides to remove the servlet (server shutdown, undeploy, reload).
- Purpose: free resources — close DB connections, stop background threads, close file handles.
- After destroy() returns, the servlet is eligible for garbage collection.

6. Garbage Collection

- JVM GC will eventually reclaim the servlet object memory.

Servlet Phase	Real-Time Example
Load	Shop is opened
Instantiate	Shopkeeper arrives
init()	Shopkeeper prepares shop
service()	Shopkeeper serves customers
destroy()	Shop closes
GC	Cleaners clean place

Http Rquest Handling

```
<html>
<body>
    <h2>Send GET Request</h2>
    <a href="api?name=Selva">Send GET Request</a>

    <hr>

    <h2>Send POST Request</h2>
    <form action="api" method="post">
        Message: <input type="text" name="message">
        <input type="submit" value="Send POST">
    </form>
</body>
</html>
```

java

```
package com.example;

import java.io.IOException;

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/api")
public class SimpleHttpServlet extends HttpServlet {

    // ========== Handle GET request ==========
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {

        String name = request.getParameter("name");  // Read data from URL

        response.setContentType("text/html");
        response.getWriter().println("<h2>GET Request Received</h2>");
        response.getWriter().println("<p>Name: " + name + "</p>");
    }

    // ========== Handle POST request ==========
    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {

        String message = request.getParameter("message"); // Read form data

        response.setContentType("text/html");
        response.getWriter().println("<h2>POST Request Received</h2>");
        response.getWriter().println("<p>Message: " + message + "</p>");
    }

    // ========== Handle PUT request ==========
    @Override
    protected void doPut(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {

        response.setContentType("text/plain");
        response.getWriter().println("PUT Request Received");
    }

    // ========== Handle DELETE request ==========
    @Override
    protected void doDelete(HttpServletRequest request, HttpServletResponse
response)
            throws IOException {

        response.setContentType("text/plain");
        response.getWriter().println("DELETE Request Received");
    }
}
```