JAVA SERVLET PROGRAMMING

Instructor: DieuNT1

Learning Goals

Understand Servlet

Can use Servlet to develop web application

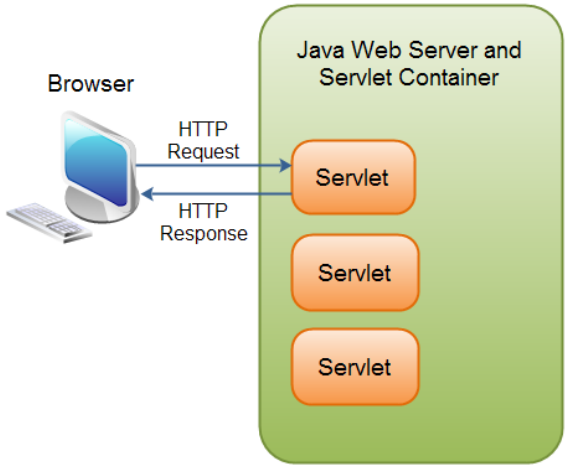
Table Content

Section 1

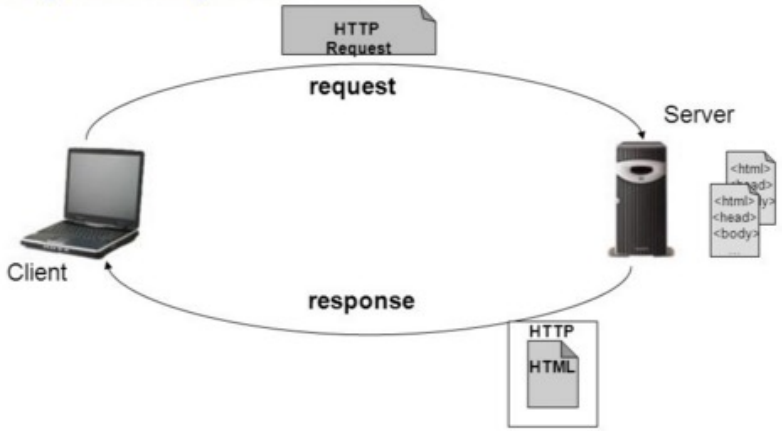
INTRODUCTION TO SERVLET

Servlet Overview

- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Performance is significantly better.
- Servlets are platform-independent because they are written in Java.

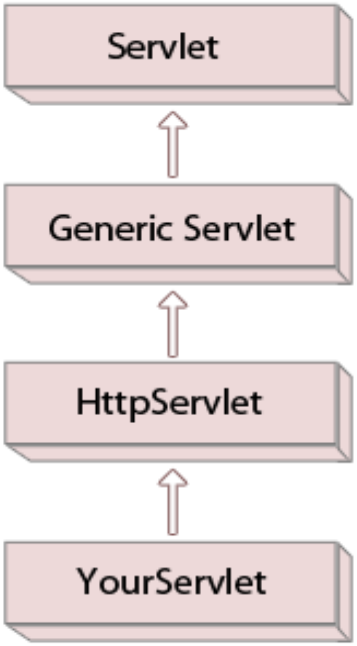


- **Request-response model.**

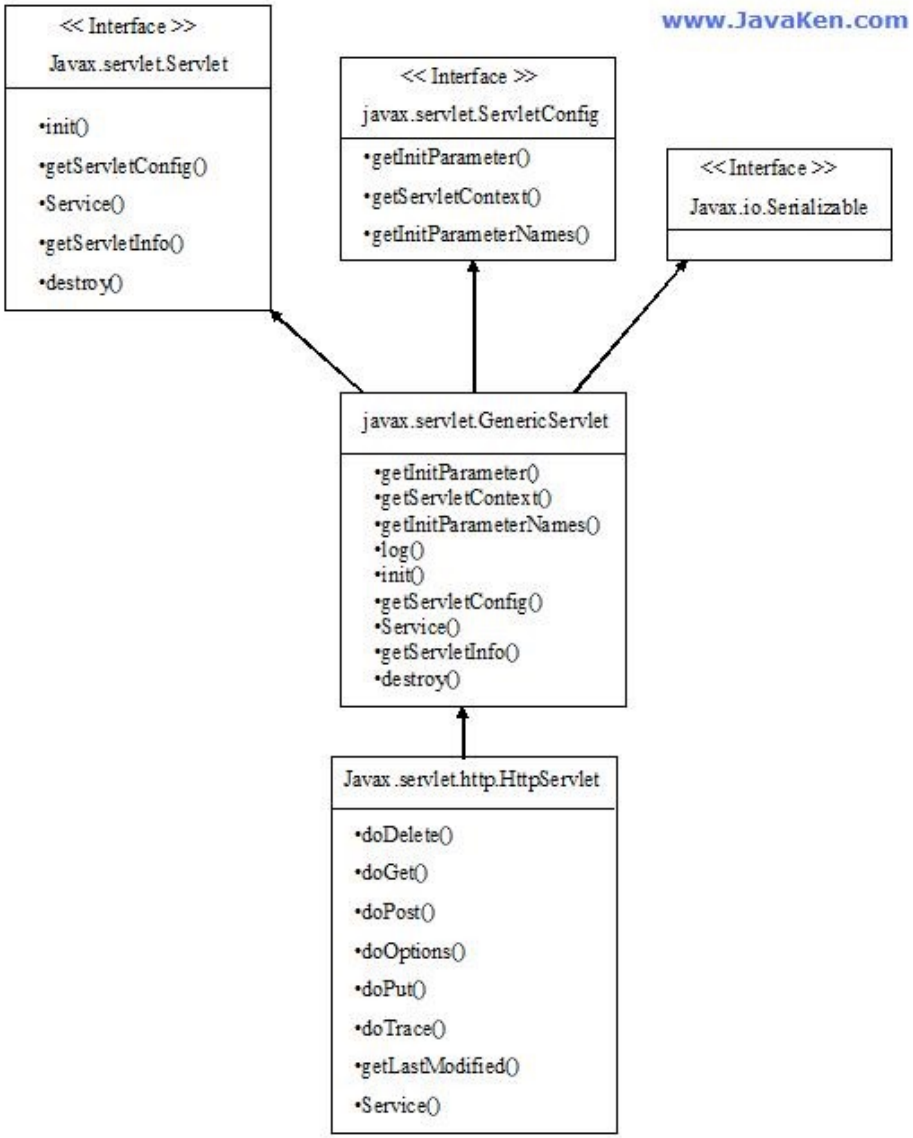


Servlet Architecture

- A **Servlet** is a class, which implements the **javax.servlet.Servlet** interface.
- To write a servlet we need to **implement Servlet** interface.
  - ✓ Servlet interface can be implemented directly or indirectly by extending **GenericServlet** or **HttpServlet** class.
  - ✓ Purpose of extending the HttpServlet class is to provide the HTTP **specific services** to **your servlet**.

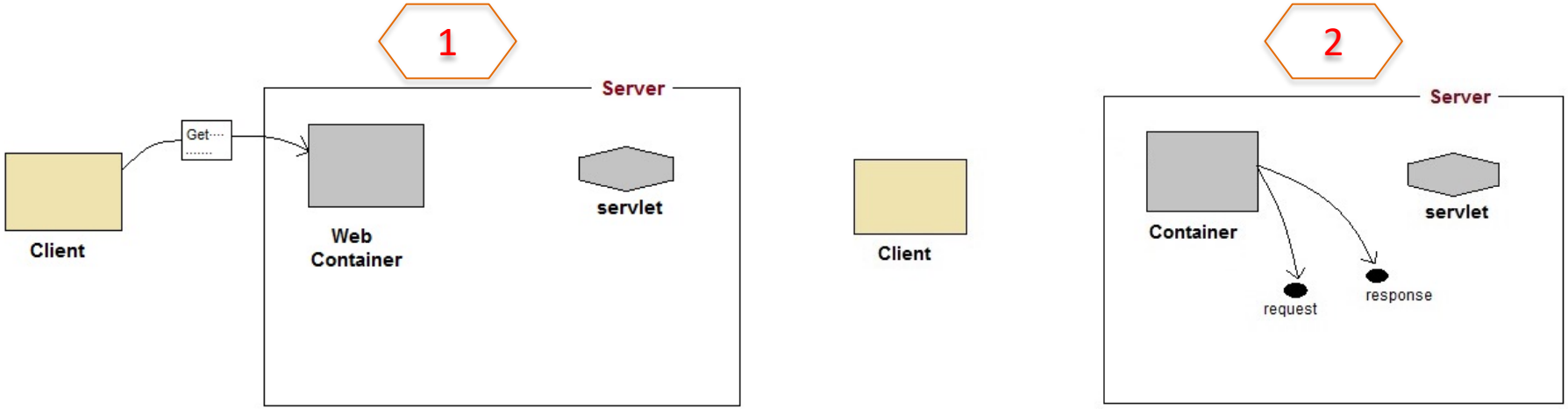


Servlet Architecture



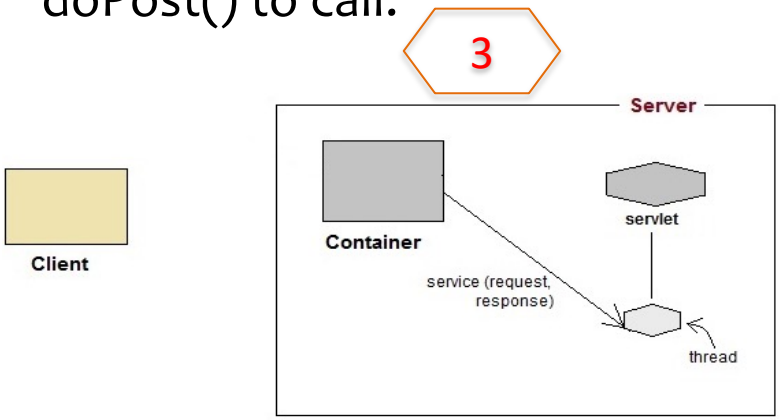
Servlet Application works

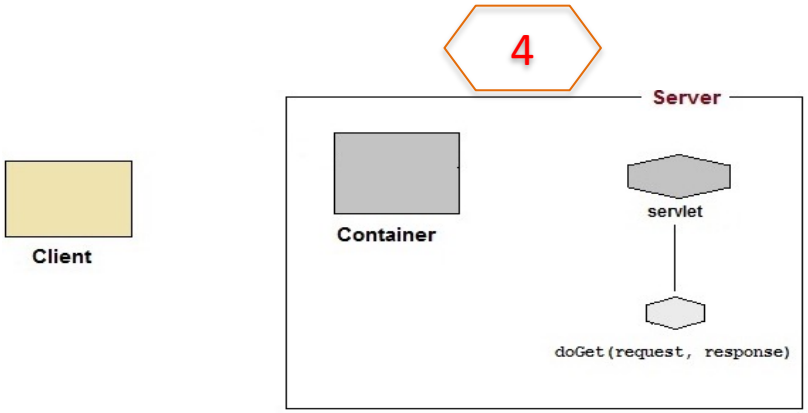
- **Web container** is responsible for managing execution of servlets and JSP pages or Java EE application:
  - ✓ User sends request for a servlet by clicking a link that has URL to a servlet.
  - ✓ The container finds the servlet using **deployment descriptor** and creates two objects:
    - **HttpServletRequest**
    - **HttpServletResponse**



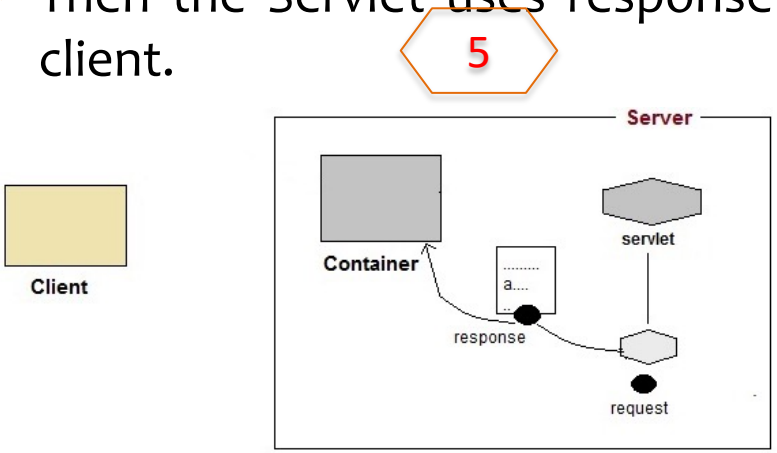
Servlet Application works

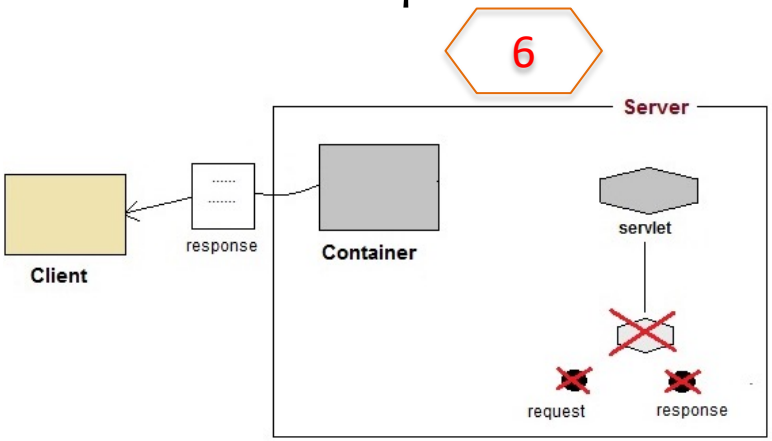
- Then the container calls the Servlet's *service*() method and passes the request, response objects as arguments.
- The service() method, then decides which servlet method, doGet() or doPost() to call.





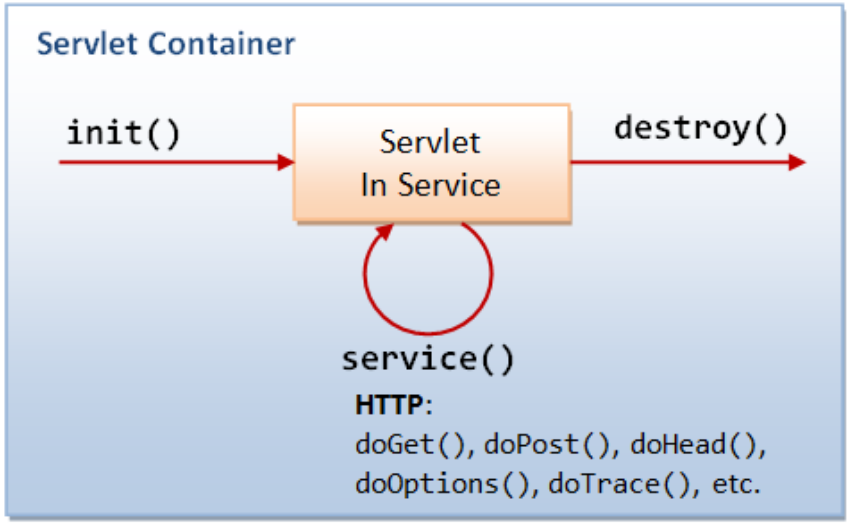
- Then the Servlet uses response object to write the response back to the client.



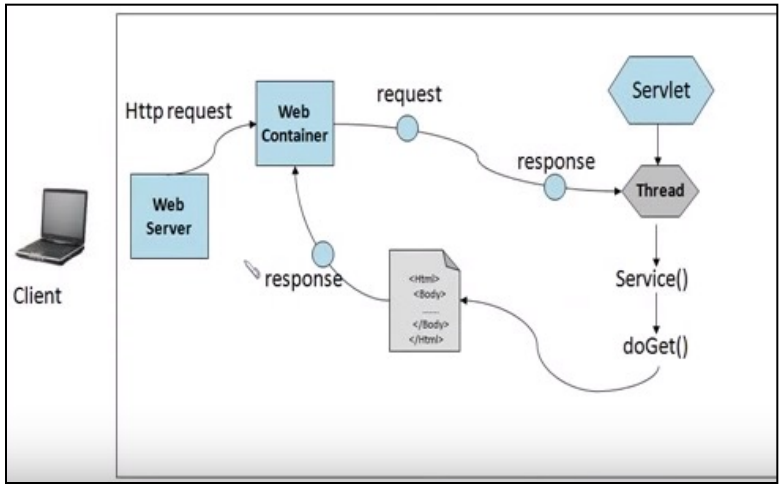


Servlet Life Cycle

- 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.



Servlet container provides the execution environment for Servlet.



The **Web container** manages a servlet by invoking various life cycle methods.

Section 2

SERVLET API

HttpServlet Class

- The **HttpServlet** class *extends* the **GenericServlet** class and implements **Serializable** interface.
  - ✓ It provides http specific methods such as *doGet*, *doPost*, *doHead*, *doTrace* etc.
- Method **`doGet`** responds to **`get`** requests
  - ✓ Retrieve the content of a URL.
- Method **`doPost`** responds to **`post`** requests
  - ✓ Post data from an HTML form to a server-side form handler.
  - ✓ Browsers cache Web pages.
- **HttpServletRequest** and **`HttpServletResponse`** objects
  - ✓ Created by the servlet container and passed as an argument to the servlet's service methods (doGet, doPost, etc.)
  - ✓ **HttpServletRequest** object contains request from the client,
  - ✓ **HttpServletResponse** object provides HTTP-specific functionality in sending a response (access HTTP headers, cookies, etc.)

HttpServlet Class

- **public void service(ServletRequest req,ServletResponse res)**:
  - ✓ dispatches the request to the protected service method by converting the request and response object into http type.
- **protected void service(HttpServletRequest req, HttpServletResponse res):**
  - ✓ receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
- **protected void doGet(HttpServletRequest req, HttpServletResponse res)**:
  - ✓ handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)**:
  - ✓ handles the POST request. It is invoked by the web container.
- **protected void doHead(HttpServletRequest req, HttpServletResponse res)**
  - ✓ handles the HEAD request. It is invoked by the web container.
- **protected void doOptions(HttpServletRequest req, HttpServletResponse res)**
  - ✓ handles the OPTIONS request. It is invoked by the web container.
- **protected void doPut(HttpServletRequest req, HttpServletResponse res)**
  - ✓ handles the PUT request. It is invoked by the web container.

HttpServlet Class

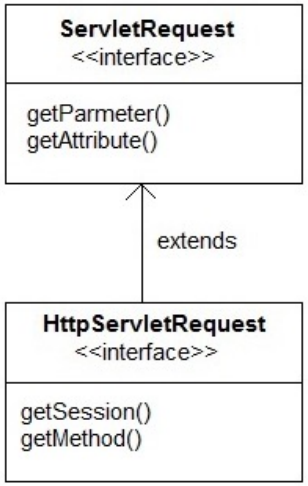
- **protected void doTrace(HttpServletRequest req, HttpServletResponse res):**
  - ✓ handles the TRACE request. It is invoked by the web container.
- **protected void doDelete(HttpServletRequest req, HttpServletResponse res)**
  - ✓ handles the DELETE request. It is invoked by the web container.
- **protected long getLastModified(HttpServletRequest req)**
  - ✓ returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Section 3

SERVLET REQUEST AND RESPONSE

HttpServletRequest

- **HttpServletRequest** interface adds the methods that relates to the **HTTP** protocol.
- Methods (extension/addition method)

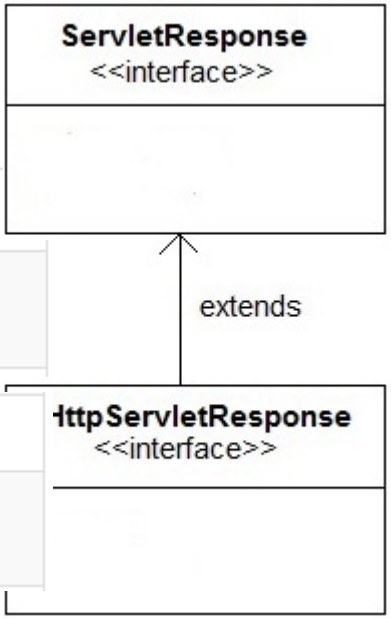


String `getParameter(String name)`	returns value of parameter by name
void `removeAttribute(String name)`	removes an attribute from this request
void `setAttribute(String name, Object o)`	stores an attribute in this request.
String[] `getParameterValues(String name)`	returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist
Cookies `getCookies()`	returns an array containing all of the Cookie objects the client sent with this request
HttpSession `getSession()`	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session

HttpServletResponse

- **HttpServletResponse** interface adds the methods that relates to the **HTTP** response.
- Methods (extension/addition method)

PrintWriter `getWriter()`	returns a PrintWriter object that can send character text to the client.
void `setContentType(String type)`	sets the content type of the response being sent to the client before sending the respond.
void `addCookie(Cookie cookie)`	adds the specified cookie to the response.
void `sendRedirect(String location)`	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer



RequestDispatcher

- **RequestDispatcher** is an interface, implementation of which defines an object which can dispatch request to any resources(such as HTML, Image, JSP, Servlet) on the server.
- **Methods:**

Methods	Description
void `forward(ServletRequest request, ServletResponse response)`	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server
void `include(ServletRequest request, ServletResponse response)`	includes the content of a resource (servlet, JSP page, HTML file) in the response

- **Example:**

ServletRequest object

resource name

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");

rs.forward(request,response);
```

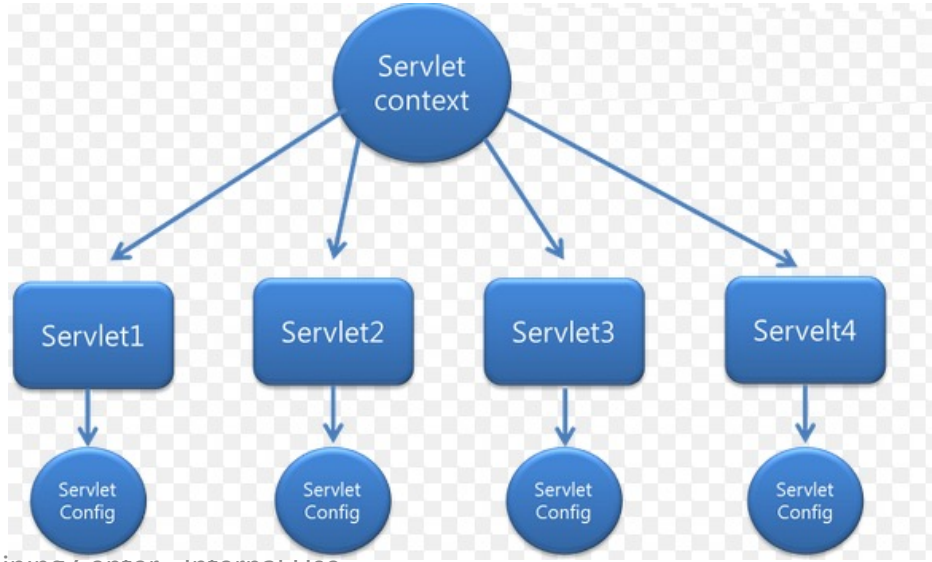
forward the request and response to "hello.html" page

RequestDispatcher

- **redirection vs request dispatching**
- The main difference between a **redirection** and a **request dispatching**:
  - ✓ redirection makes the client(browser) create a new request to get to the resource, the user can see the new URL;
  - ✓ request dispatch get the resource in same request and URL does not changes.

ServletContext

- An object of ServletContext is created by the web container at time of deploying the project. This object can be used to **get configuration information from web.xml file**.
- Servlet Context has 3 main methods:
  - ✓ GetAttribute ()
  - ✓ SetAttribute ()
  - ✓ RemoveAttribute ()
- Servlet Context help provides communication between the servlet
- Servlet Context can also be used to obtain configuration information web.xml.



ServletContext Example

web.xml

1

```
<display-name>JAVASERVLET</display-name>

<welcome-file-list>
    <welcome-file>Login</welcome-file>
</welcome-file-list>

<context-param>
    <param-name>USER_NAME</param-name>
    <param-value>admin</param-value>
</context-param>

<context-param>
    <param-name>PASSWORD</param-name>
    <param-value>admin123</param-value>
</context-param>
```

web.xml | LoginServlet.java

2

```
 * Servlet implementation class LoginServlet
 */
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private static String USER_NAME;
    private static String PASSWORD;


    public void init(ServletConfig config) throws ServletException {

        System.out.println("LoginServlet::init::BEGIN");

        ServletContext context = config.getServletContext();
        USER_NAME = context.getInitParameter("USER_NAME");
        PASSWORD  = context.getInitParameter("PASSWORD");

        System.out.println("LoginServlet::init::END");

    }
```

Servlet Context
Sample

- We can change the init() method of the SurveyServlet as below:

```
// set up database connection and prepare SQL statements
public void init( ServletConfig config ) throws ServletException
{
   String dbDriver, dbURL;
   ServletContext context = config.getServletContext();

   dbDriver = context.getInitParameter("DB_Driver");
   dbURL = context.getInitParameter("DB_URL");

   // attempt database connection and create PreparedStatements
   // ...
}
```

- Then we need to amend the web.xml file to specify the initial context parameters:

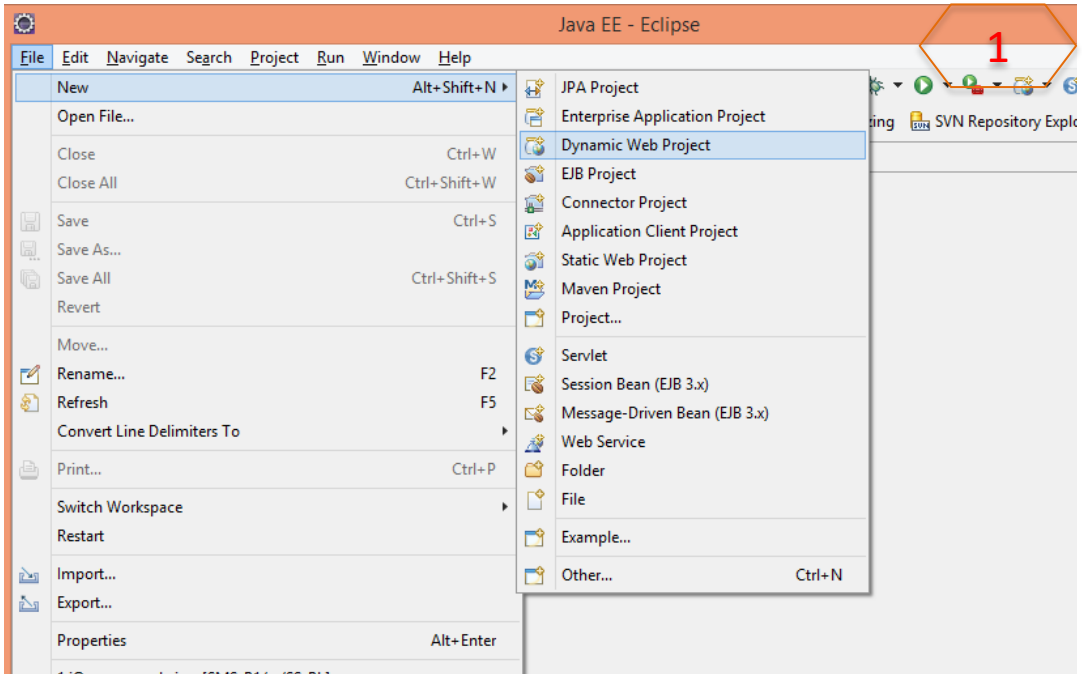
```
<context-param>
    <param-name>DB_URL</param-name>
    <param-value>jdbc:mysql://localhost:3306/test</param-value>
</context-param>

<context-param>
    <param-name>DB_Driver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
</context-param>
```

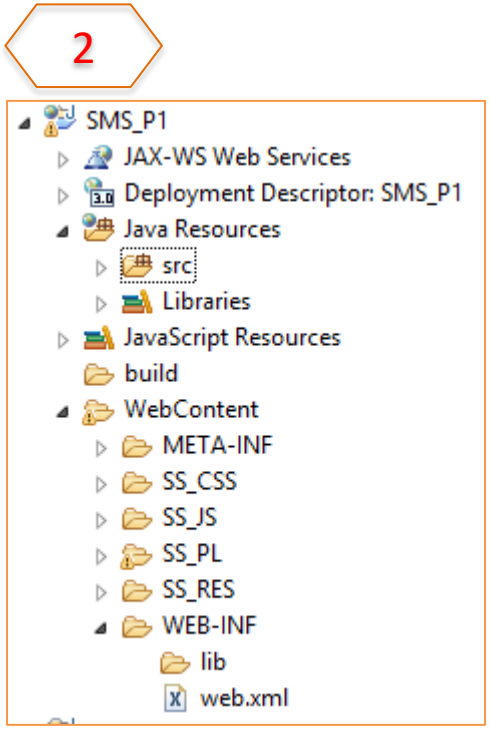
Section 4

CREATE SERVLET IN ECLIPSE IDE

Steps to create Servlet using Eclipse IDE

- Create a **Dynamic Web Project,** give a name to your project (Ex: **SMS_P1**)

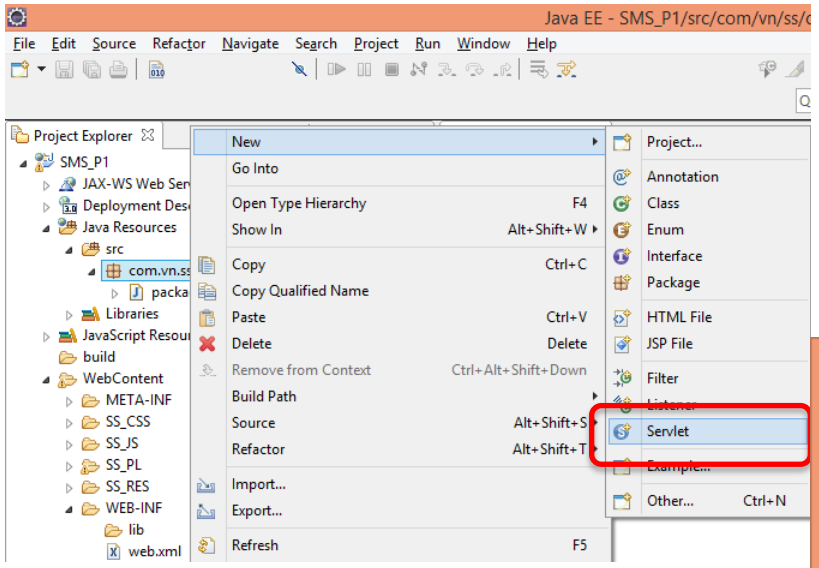




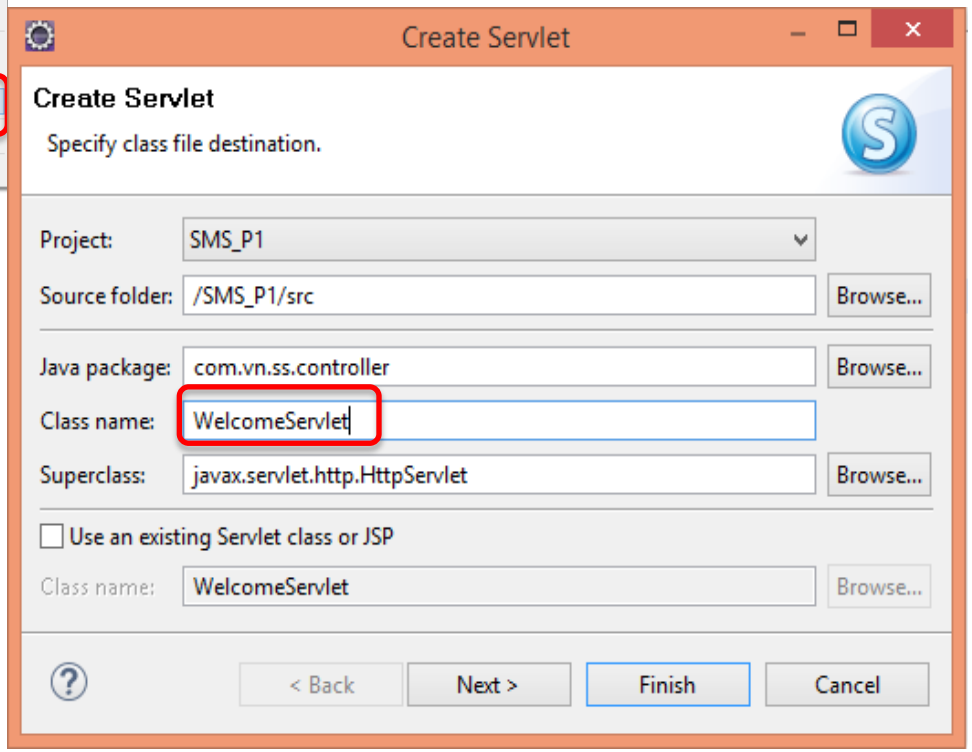
*A **directory structure** of your Project will be automatically created by Eclipse IDE.*

Steps to create Servlet using Eclipse IDE

- Create a **Servlet** class by extends HttpServlet.



- Add **servlet-api.jar** JAR file to your project



Handling HTTP get Requests

- Create a Servlet (*WelcomeServlet.java*):

```
package atjb.day3;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class WelcomeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletExc
        // TODO Auto-generated method stub
        System.out.println("Hello");
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // send XHTML page to client
        // start XHTML document
        out.println("<?xml version = \"1.0\"?>");

        out.println("<!DOCTYPE html PUBLIC \"-//W3C//DTD "
                + "XHTML 1.0 Strict//EN\" \"http://www.w3.o
                + "/TR/xhtml1/DTD/xhtml1-strict.dtd\">");
        out.println("<html xmlns = \"http://www.w3.org/1999
        // head section of document
        out.println("<head>");
        out.println("<title>A Simple Servlet Example</title>");
        out.println("</head>");

        // body section of document
        out.println("<body>");
        out.println("<h1>Welcome to Servlets!</h1>");
        out.println("</body>");

        // end XHTML document
        out.println("</html>");
        out.close(); // close stream to complete the page
    }
}
```

Import the **javax.servlet** and **javax.servlet.http** packages.

Extends **HttpServlet** to handle HTTP **get** requests and HTTP **post** requests.

Override method **doGet** to provide custom get request processing.

Uses the **response** object's **setContentType** method to specify the content type of the data to be sent as the response to the client.

Uses the **response** object's **getWriter** method to obtain a reference to the **PrintWriter** object that enables the servlet to send content to the client.

Create the XHTML document by writing strings with the **out** object's **println** method.

Closes the output stream, flushes the output buffer and sends the information to the client.

Create Deployement Descriptor

```
1   <!DOCTYPE web-app PUBLIC
2      "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3      "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
4
5   <web-app>
6
7      <!-- General description
8      <display-name>
9         Sample
10        Servlet Examples
11     </display-name>
12
13     <description>
14        This is the Web application in which we
15        demonstrate our JSP and Servlet examples.
16     </description>
17
18     <!-- Servlet definition
19     <servlet>
20        <servlet-name>welcome</servlet-name>
21
22        <description>
23           A simple servlet that handles an HTTP get request.
24        </description>
25
26        <servlet-class>
27           atjb.day3.WelcomeServlet
28        </servlet-class>
29     </servlet>
30
```

Deployment descriptor (web.xml) for the Web application.

Element **web-app** defines the configuration of each servlet in the Web application and the servlet mapping for each servlet.

Element **display-name** specifies a name that can be displayed to the administrator of the server on which the Web application is installed.

Element **description** specifies a description of the Web application that might be displayed to the administrator of the server.

Element **servlet** describes a servlet.

Element **servlet-name** is the name for the servlet.

Element **description** specifies a description for this particular servlet.

Element **servlet-class** specifies compiled servlet's fully qualified class name.

Create Deployement Descriptor

```
31      <!-- Servlet mappings -->
32      <servlet-mapping>
33         <servlet-name>welcome</servlet-name>
34         <url-pattern>/welcomeServlet</url-pattern>
35      </servlet-mapping>
36
37   </web-app>
```

Element **servlet-mapping** specifies **servlet-name** and **url-pattern** elements.

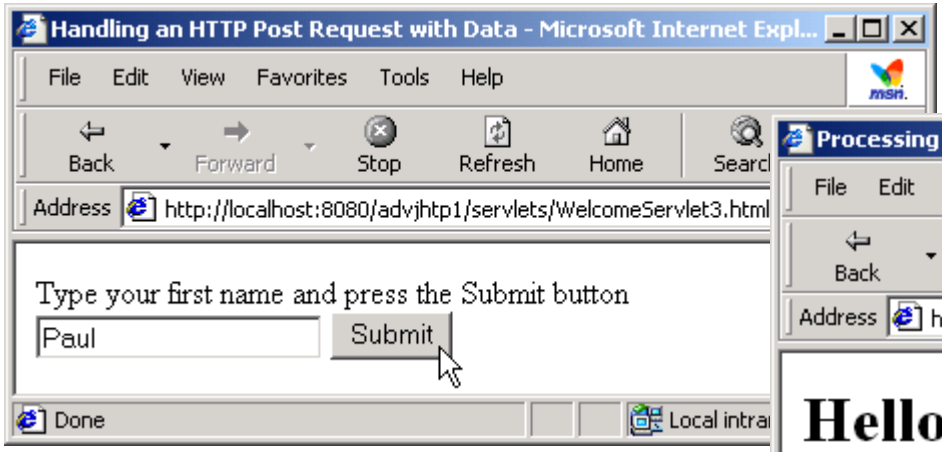
Handling HTTP get Requests

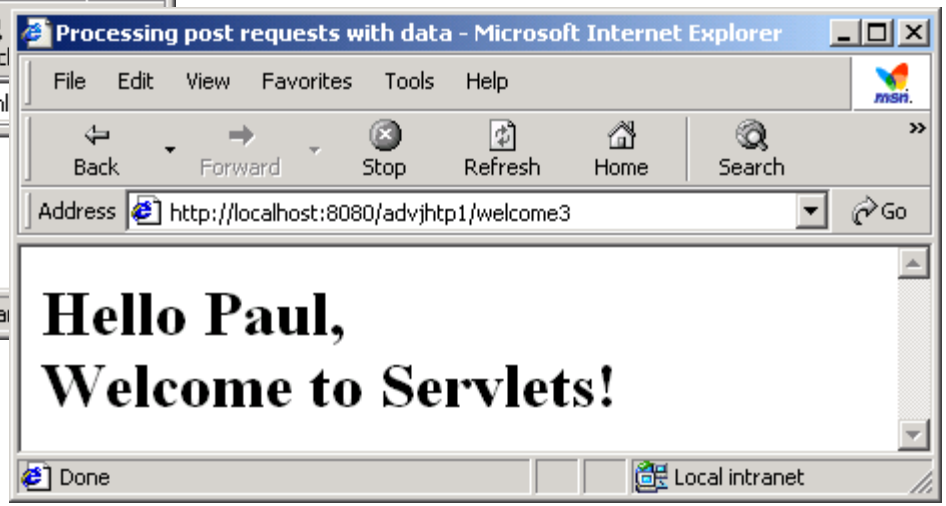
- Jsp page in which the form's action invokes **WelcomeServlet** through the alias **welcome** specified in **web.xml.**
- **Example: Welcome.jsp**

```
1  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2      pageEncoding="ISO-8859-1"%>
3  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Handling an HTTP Get Request</title>
8  </head>
9  <body>
10     <form action="/WelcomeServlet" method="get">
11         <p>
12             <label>Click the button to invoke the servlet
13                 <input type="submit" value="Get HTML Document" />
14             </label>
15         </p>
16     </form>
17 </body>
18 </html>
19
```

Handling HTTP post Requests

- HTTP post request
  - ✓ Post data from an HTML form to a server-side form handler
  - ✓ Browsers cache Web pages
- Practices time:
  - ✓ *Let's try to see differences between HTTP post and HTTP get requests*
  - ✓ *Use HTTPServletRequest's getParameter method() to get param value*

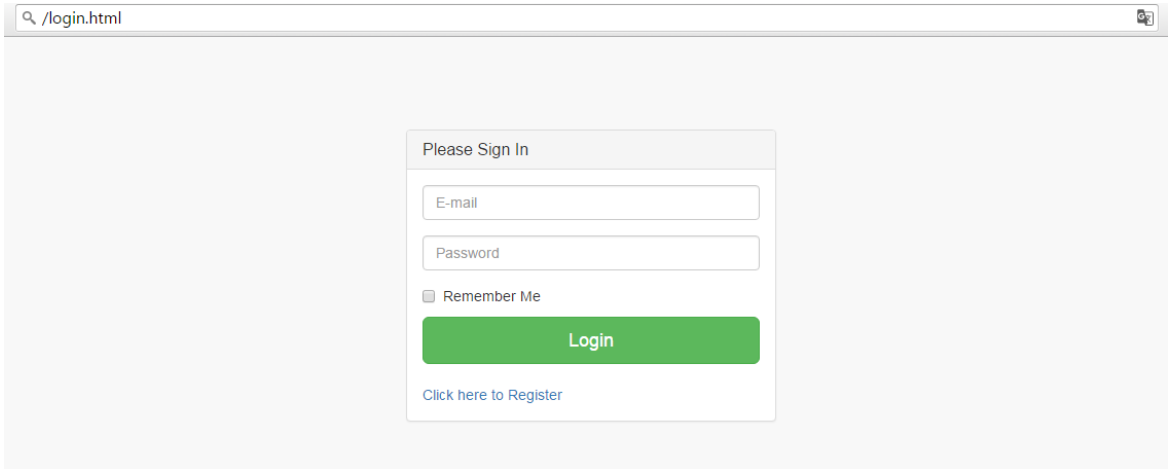


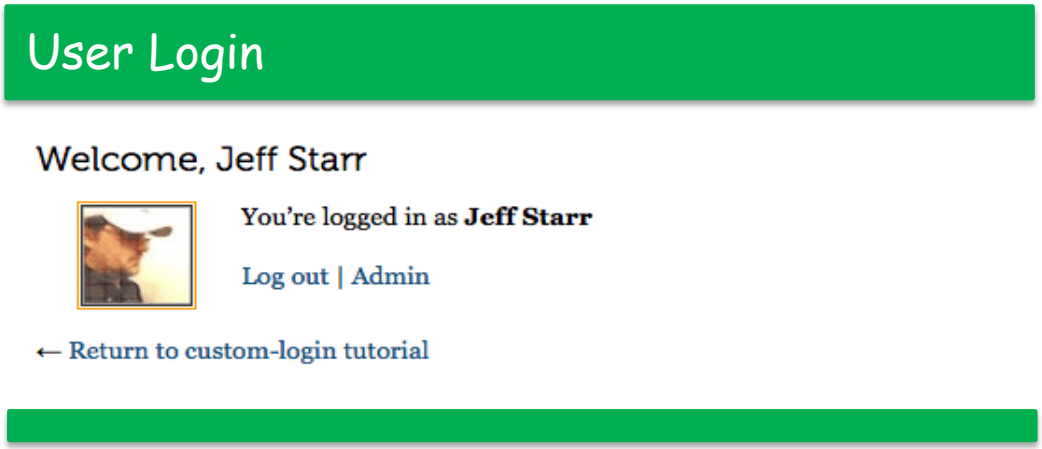


Practices time

❖ Handling HTTP post Requests sample:

```
<form action="${pageContext.request.contextPath}/WelcomeServlet?"
                                                    method="post">
```





Summary

- Introduction to Servlet
- Servlet API
- Servlet Request and Response
- Servlet Context
- Create servlet in Eclipse IDE

Thank you