

SERVLETS

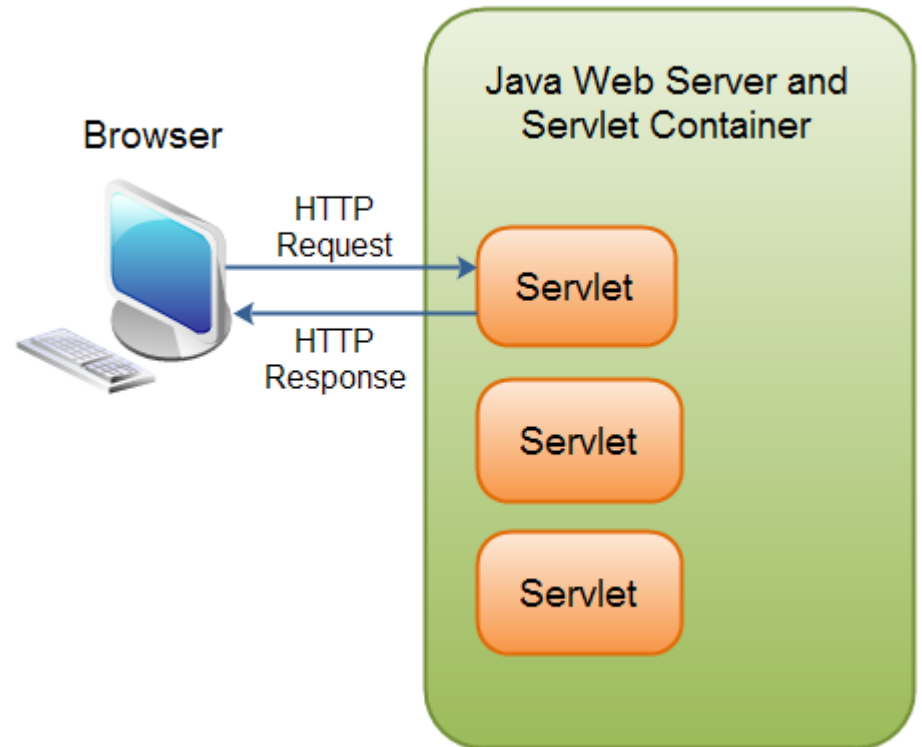


Introduction

- ❑ Need for creating Dynamic Web Pages.
- ❑ Java Servlets are used to design dynamic web pages.
- ❑ Servlets are the Java programs that runs on the Java-enabled web server or application server.
- ❑ They are used to handle the request obtained from the web server, process the request, produce the response, then send response back to the web server.

Introduction

- Java Servlets 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.



Common Gateway Interface

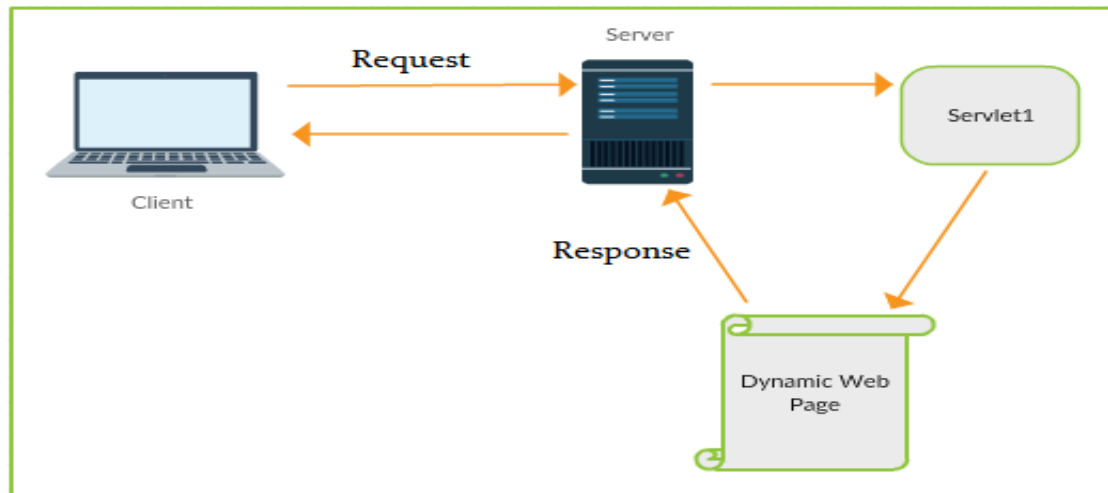
- ❑ In client- server computing, each application had its own client program and it worked as a user interface.
- ❑ It was supposed to be installed on each user's personal computer.
- ❑ The Common Gateway Interface (CGI) was developed for creating dynamic content.
- ❑ By using the CGI, a web server passes requests to an external program and after executing the program the content is sent to the client as the output.

Common Gateway Interface

- ❑ CGI creates a new process when a server receives a request.
- ❑ CGI creates a process for each request.
- ❑ This required significant server resources and time.
- ❑ Limited number of requests that can be processed concurrently.
- ❑ CGI applications are platform dependent.

Servlets

- 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.



Servlets

- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Java Servlets often serve the same purpose as Common Gateway Interface (CGI) but offer several advantages.

Servlets vs CGI

SERVLETS	CGI
Platform Independent	Platform Dependent
Every request is handled by a lightweight Java Thread.	Every request is handled by a heavyweight Process.
Java security manager on the server protects the resources on a server machine.	No support for Security Manager
Availability of complete Java class libraries	Limited Libraries
Servlets can share data among each other	CGI does not provide sharing property
Servlets can perform session tracking	CGI cannot perform session tracking
Servlets can read and set HTTP headers, handle cookies	CGI can't read and set HTTP headers, handle cookies

Servlet Applications

- ❑ Search engines
- ❑ E-commerce applications
- ❑ Shopping carts
- ❑ Product catalogs
- ❑ Intranet application
- ❑ Groupware applications: bulletin boards, file sharing, etc.

Servlet Life Cycle

1. Load Servlet Class.
2. Create Instance of Servlet.
3. Call the servlets `init()` method.
4. Call the servlets `service()` method.
5. Call the servlets `destroy()` method.

Servlet Life Cycle

- Browser sends a HTTP request to the web server (through URL, submitting a form, etc)
- Web server receives the request and maps this request to a particular Servlet.
 - ▣ The servlet is retrieved and dynamically loaded into the server address space.
- `init()` method of the servlet is called by the web server only when the servlet is first loaded into the memory.
`public void init() throws ServletException {`
`// Initialization code...`
`}`

Servlet Life Cycle

- Web server invokes service() method of the servlet.
 - ▣ This method is called to process the HTTP request.
 - ▣ Read the parameters from the request and send a response back to the browser.
 - ▣ The servlet remains in the server's address space and is available to process any other HTTP requests received from clients.
 - ▣ Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
 - ▣ The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Servlet Life Cycle

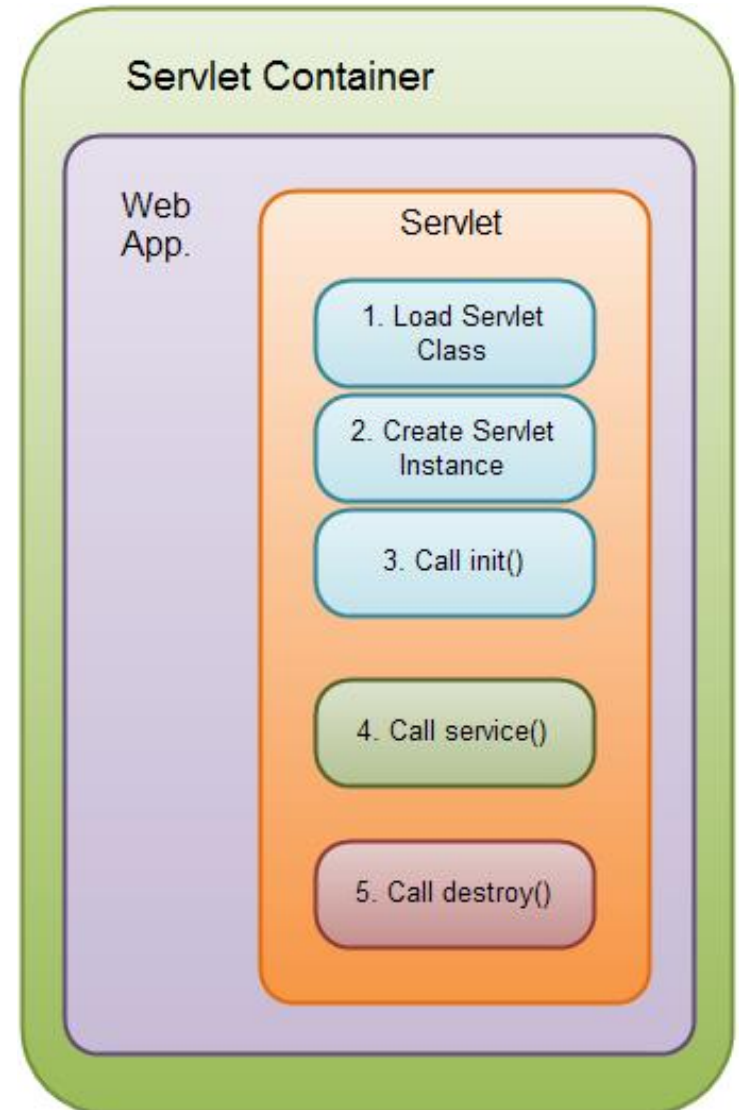
- Signature of service()

```
public void service(ServletRequest request,  
ServletResponse response)  
throws ServletException, IOException{ }
```
- The server decides to unload the servlet from its memory and calls the destroy() method to relinquish any resources.

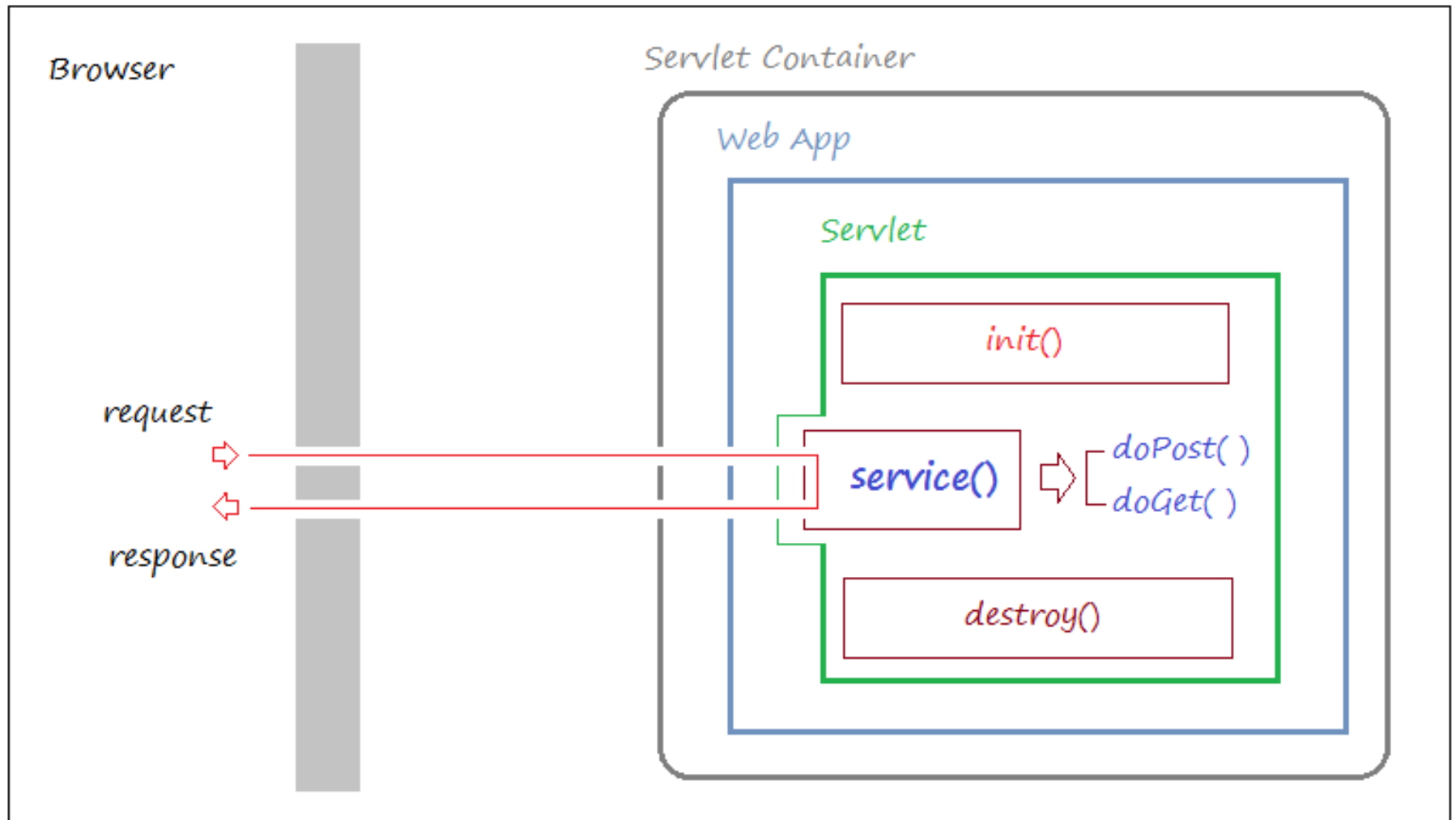
```
public void destroy() {  
// Finalization code...  
}
```

Servlet Life Cycle

- When HTTP calls for a Servlet
 - ▣ Not loaded: Load, Create, Init, Service
 - ▣ Already loaded: Service



Servlet Life Cycle



Servlet Container

- Servlet container is the component of a web server that interacts with Java servlets.
- A Servlet container is responsible for
 - ▣ Managing the lifecycle of servlets.
 - ▣ Mapping a URL to a particular servlet
 - ▣ Ensuring that the URL requester has the correct access rights.
- A web container implements the web component contract of the Java EE architecture, specifying a runtime environment for web components that includes security, concurrency, lifecycle management, transaction, deployment, and other services.

doGet() Method

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,  
    HttpServletResponse response)  
    throws ServletException, IOException {  
    // Servlet code  
}
```

doPost() Method

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,  
    HttpServletResponse response)  
    throws ServletException, IOException {  
    // Servlet code  
}
```

Generic Servlet vs HTTP servlet

- There are two different kinds of servlet classes.
 1. **GenericServlet** is just that, a generic, protocol-independent servlet.
 2. **HttpServlet** is a servlet tied specifically to the HTTP protocol.

Generic Servlet vs HTTP servlet

GenericServlet	HttpServlet
Defines a generic, protocol-independent servlet.	Defines a HTTP protocol specific servlet.
Gives a blueprint and makes writing servlet easier.	Gives a blueprint for Http servlet and makes writing them easier.
provides simple versions of the lifecycle methods init, service, and destroy and of the methods in the ServletConfig interface	extends the GenericServlet and hence inherits the properties of GenericServlet.

Servlet Libraries

- Two packages contain the classes and interfaces that are required to build servlets. These are
 1. `javax.servlet`, and
 2. `javax.servlet.http`.
- They constitute the Servlet API.
- These packages are not part of the Java core packages.
- Instead, they are standard extensions provided by the web servers.

javax.servlet package

Interfaces	Description
Servlet	Declares life cycle methods for a Servlet
ServletConfig	Allows Servlets to get initialization parameters
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request
ServletResponse	Used to write data to a client response

javax.servlet package

Classes	Description
GenericServlet	Implements the Servlet and ServletConfig interfaces.
ServletInputStream	Provides an input stream for reading requests from a client
ServletOutputStream	Provides an output stream for writing responses to a client
ServletException	Indicates a Servlet error occurred.
UnavailableException	Indicates a Servlet is unavailable.

javax.servlet.http package

Interfaces	Description
HttpServletRequest	Enables servlets to read data from an HTTP request
HttpServletResponse	Enables servlets to write data to an HTTP response
HttpSession	Allows session data to be read and written
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session

javax.servlet.http package

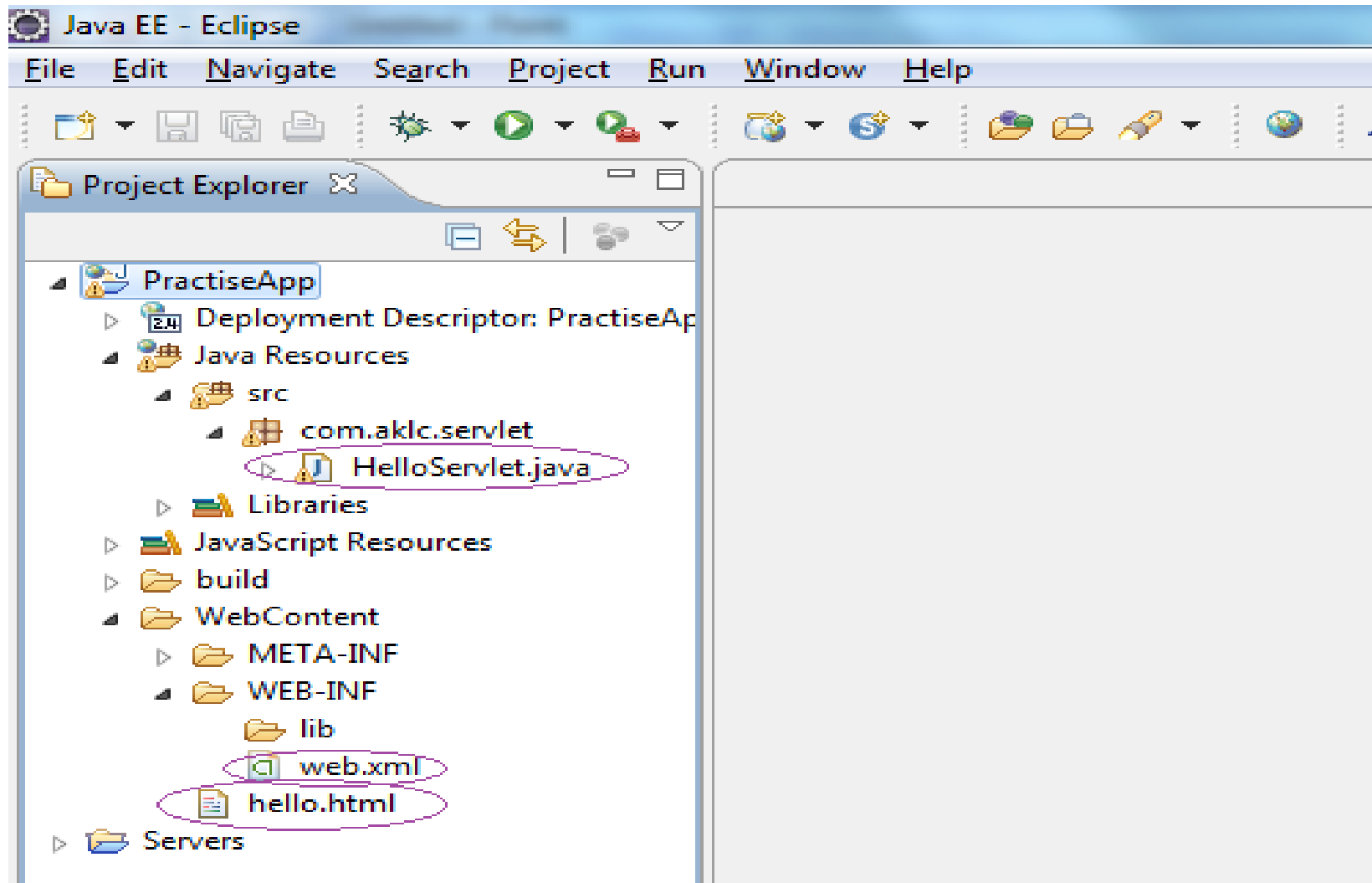
Classes	Description
Cookie	Allows state information to be stored on a client machine
HttpServlet	Provides methods to handle HTTP requests and responses
HttpSessionEvent	Encapsulates a session-changed event
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value, or that a session attribute changed

Servlet Example

□ **HelloServlet Example:**

1. **HelloServlet.java** : A servlet, which is a Java class file
2. **hello.html** : A web page to invoke the servlet by sending a request
3. **web.xml** : A deployment descriptor which will map the request from the web page to the servlet

Servlet Example



Servlet Example : HelloServlet.java

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

public class HelloServlet extends GenericServlet {
    public void service(ServletRequest req, ServletResponse resp) throws
        ServletException, IOException {

        // Set the response type (MIME Type)
        resp.setContentType("text/html");

        // So you can write to the browser (client) using this writer object
        PrintWriter pw = resp.getWriter();

        // Enclose your response string inside println() method
        pw.println("<h1 style='color:blue; background-color:yellow;'> Hello World,
            Welcome to the world of Servlet</h1>");

        // Close the writer object pw.close();
    }
}
```

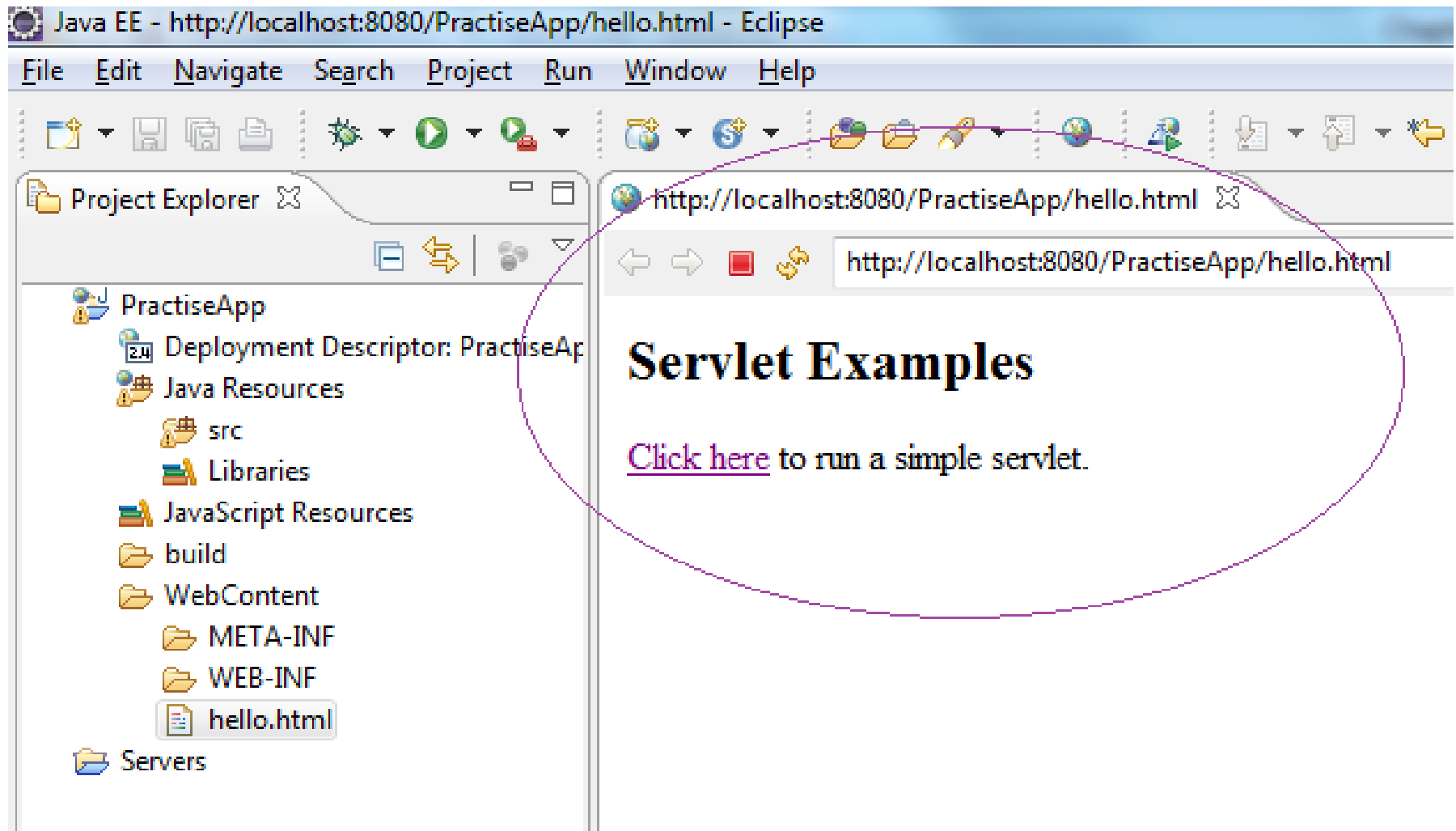
Servlet Example : hello.html

```
<html>
  <body>
    <h2> Servlet Examples </h2>
    <a href='hello'>Click here</a> to run a simple servlet.
  </body>
</html>
```

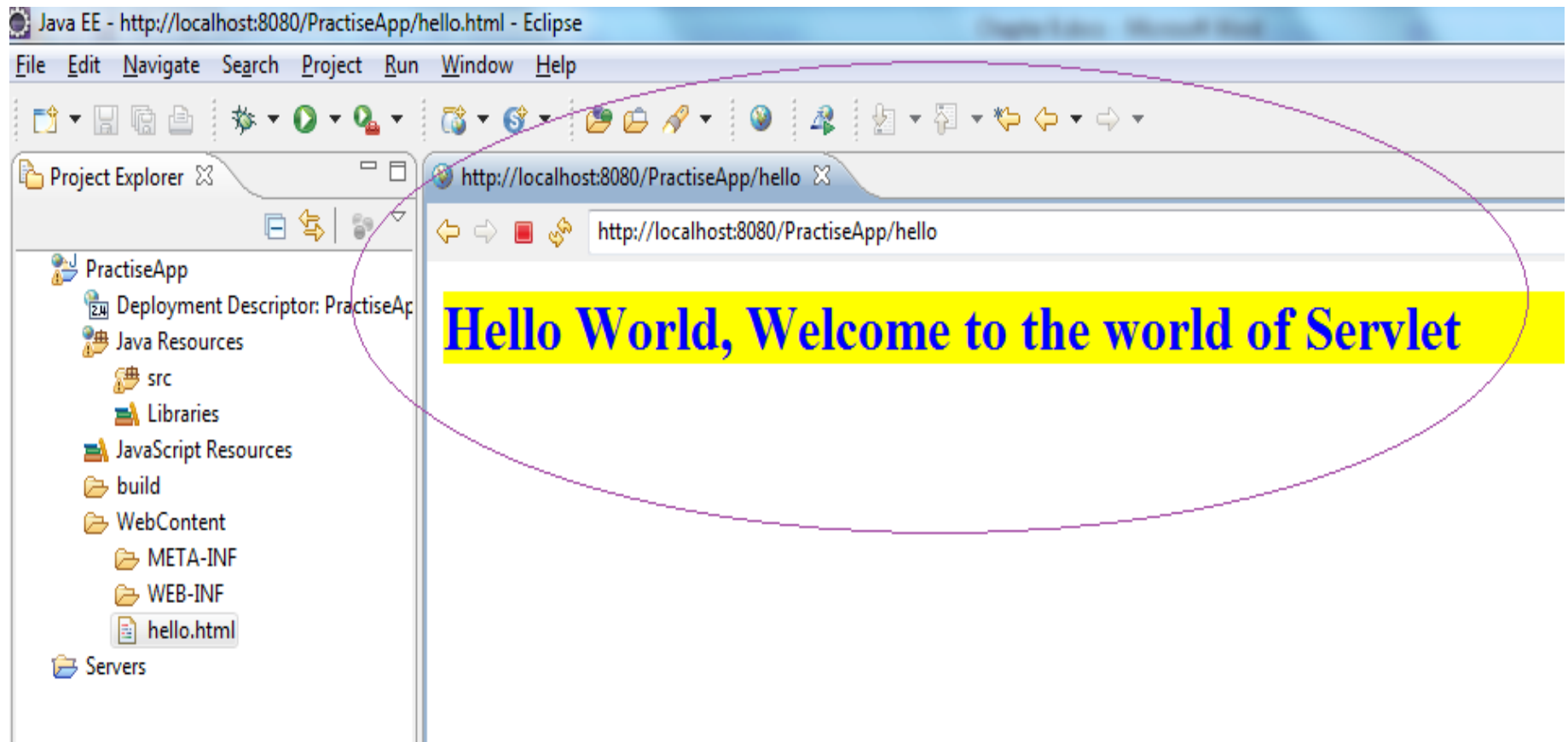
Servlet Example : web.xml

```
<servlet>  
    <servlet-name>FirstServlet</servlet-name>  
    <servlet-class>com.example.servlet.HelloServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>FirstServlet</servlet-name>  
    <url-pattern>/hello</url-pattern>  
</servlet-mapping>
```

Servlet Example : Run hello.html on the server



Servlet Example : Run hello.html on the server



HTML Markup from Servlet

Generated Markup	HelloWorld.java
<code><html></code>	<code>out.println("<html>");</code>
<code><head></code>	<code>out.println("<head>");</code>
<code><title>Hello World!</title></code>	<code>out.println("<title>Hello World!</title>");</code>
<code></head></code>	<code>out.println("</head>");</code>
<code><body></code>	<code>out.println("</head>");</code>
<code><h1>Hello World!</h1></code>	<code>out.println("<h1>Hello World!</h1>");</code>
<code></body></code>	<code>out.println("</body>");</code>
<code></html></code>	<code>out.println("</html>");</code>

Form Data

- Basics of HTML forms
- **Use the FORM element to create an HTML form**
 - ▣ ACTION attribute to designate the address of the servlet or JSP page that will process the results.
 - ▣ **<FORM ACTION="...">...</FORM>**
 - ▣ If ACTION is omitted, the data is submitted to the URL of the current page.
- **Use input elements to collect user data.**
 - ▣ Place the elements between the start and end tags of the FORM element and give each input element a NAME.
 - ▣ **<INPUT TYPE="TEXT" NAME="...">**

Form Data

- ❑ **Place a submit button near the bottom of the form.**
 - ❑ **<INPUT TYPE="SUBMIT">**
 - ❑ When the button is pressed, the URL designated by the form's ACTION is invoked.
 - ❑ With GET requests, a question mark and name/value pairs are attached to the end of the URL, where the names come from the NAME attributes in the HTML input elements and the values come from the end user.
 - ❑ With POST requests, the same data is sent, but on a separate request line instead of attached to the URL.

Reading Form Data from Servlets

□ **getParameter ()**

▣ Reading Single Values

- ▣ To read a request (form) parameter, you simply call the `getParameter` method of `HttpServletRequest`, supplying the case-sensitive parameter name as an argument.
- ▣ Supply the parameter name exactly as it appeared in the HTML source code.
- ▣ The servlet knows which request method the client used and automatically uses the appropriate method to read the data.

Reading Form Data from Servlets

□ **getParameter ()**

- ▣ Returns **Empty String** if the parameter exists but has no value (i.e., the user left the corresponding textfield empty when submitting the form).
- ▣ Returns **Null** if there is no such parameter.
- ▣ Returns **String** which corresponds to the parameter.
- ▣ Parameter names are case sensitive so, `request.getParameter("Param1")` and `request.getParameter("param1")` are not interchangeable.

Reading Form Data from Servlets

□ **getParameter ()**

- ▣ Named Field values HTML FORM

```
<INPUT TYPE="TEXT" NAME="FName">
```

- ▣ In Servlet

```
String name = request.getParameter("FName");
```

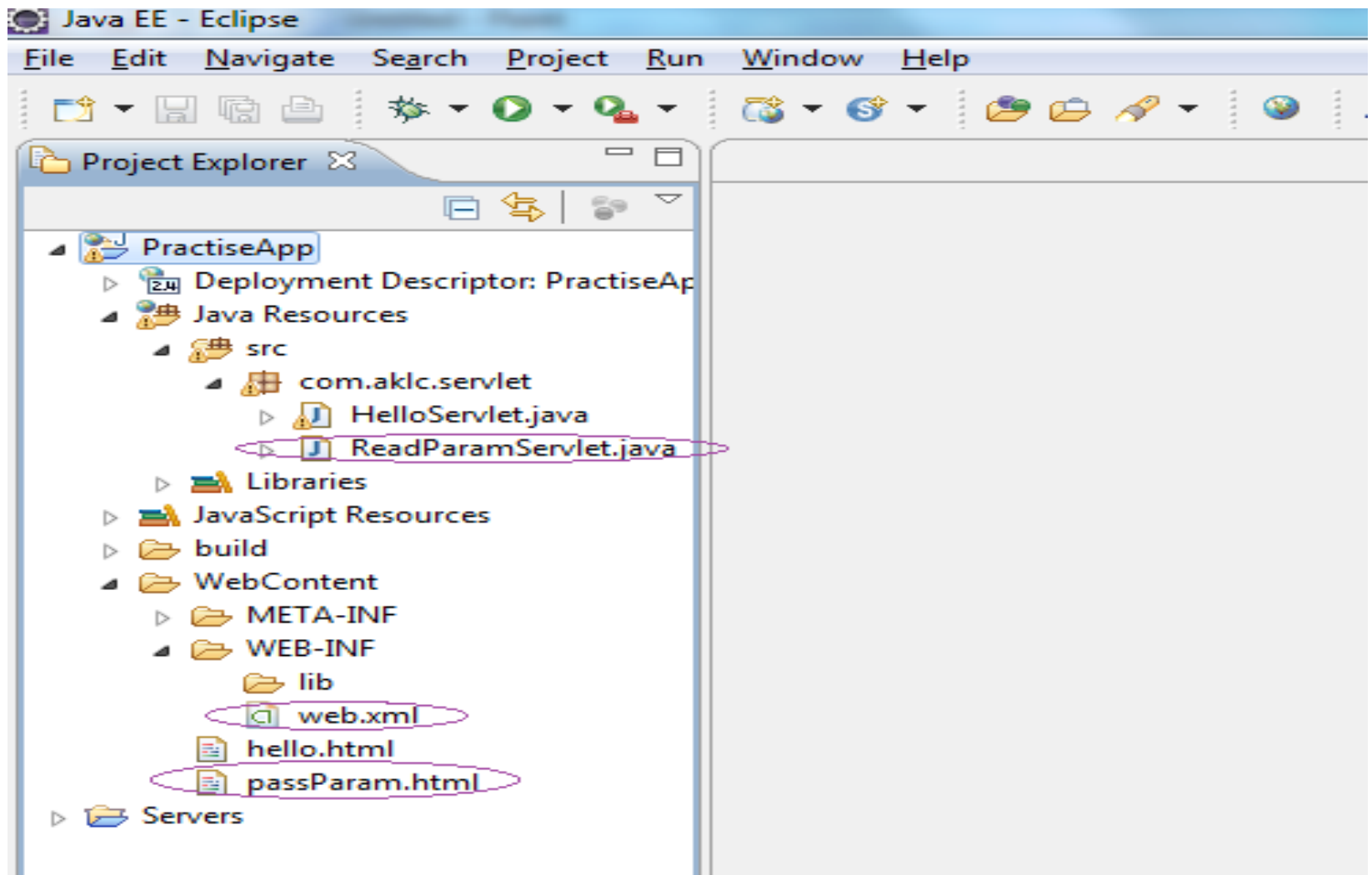


Reading request parameters

□ Example

- ▣ **ReadParamServlet.java** : A servlet, which is a Java class file. It reads the parameters from the request.
- ▣ **passParam.html** : A web page to invoke the servlet by sending a request. It passes the parameters to the servlet.
- ▣ **web.xml** : A deployment descriptor which will map the request from the web page to the servlet
Parameter Values are the string arrays as returned by `getParameterNames`.

Reading request parameters



Reading request parameters

```
//ReadParamServlet .java
```

```
import java.io.*;
import javax.servlet.*;
public class ReadParamServlet extends GenericServlet
{
    public void service(ServletRequest req, ServletResponse resp)
                        throws ServletException, IOException
    {
        resp.setContentType("text/html");
        PrintWriter pw = resp.getWriter();
        String name=req.getParameter("username");
        pw.println("<h2> Welcome, "+name+"</h2>");
        pw.close();
    }
}
```

Reading request parameters

passParam.html

```
<html>
  <body>
    Passing parameters to a Servlet
    <form action='paramEx'>
      Enter your name: <input type='text' name='username' />
      <br/> <br/>
      <input type='submit' value='Click me' />
    </form>
  </body>
</html>
```

Reading request parameters

web.xml

<servlet>

 <servlet-name>ParamServlet</servlet-name>

 <servlet-class>com.aklc.servlet.ReadParamServlet</servlet-class>

</servlet>

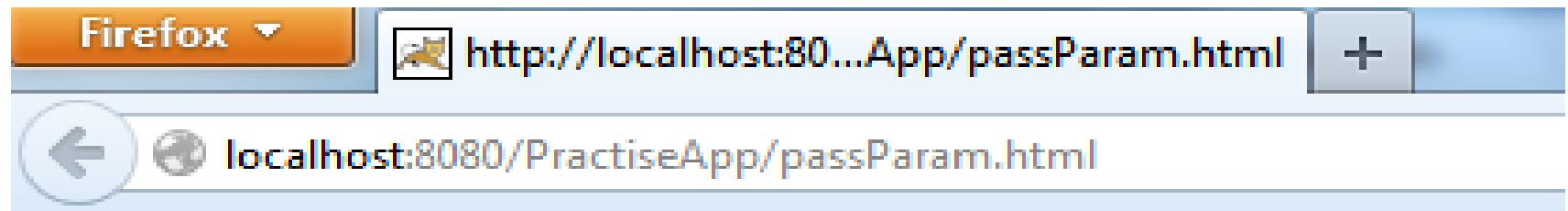
<servlet-mapping>

 <servlet-name>ParamServlet</servlet-name>

 <url-pattern>/paramEx</url-pattern>

</servlet-mapping>



Reading request parameters



Passing parameters to a Servlet

Enter your name:

Reading request parameters

  localhost:8080/PractiseApp/paramEx?username=ABC

Welcome, ABC

Reading RadioButton Data - Servlets

```
<html>
<head>
<title>Radio button example</title>
</head>
<body>
<form method="get" action="RadioButton">
<h2>Select your favorite dish</h2>
<p><input type="radio" name="dish" value="Indian"> Indian</input></p>
<p><input type="radio" name="dish" value="Continental">
Continental</input></p>
<p><input type="radio" name="dish" value="Chinese"> Chinese</input></p>
<p><input type="radio" name="dish" value="Italian"> Italian</input></p>
<p><input type="radio" name="dish" value="Russian"> Russian</input></p>
<input type="submit" value="Submit"></form>
</body>
</html>
```

Reading RadioButton Data - Servlets

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

public class RadioButton extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String value;
        res.setContentType("text/html");
        value = req.getParameter("dish");
        PrintWriter pw = res.getWriter();
        pw.println("<html><body bgcolor=green>");
        pw.println("<h3>Your favorite dish is = <font color=yellow>" + value
            + "</font></h3>");
    }
}
```

Reading Form Data from Servlets

□ **getParameterValues()**

- If the same parameter name might appear in the form data more than once, you should call `getParameterValues` (which returns an array of strings) instead of `getParameter`.
- The return value of `getParameterValues` is null for nonexistent parameter names and is a one-element array when the parameter has only a single value.
- Best to ensure that each textfield, checkbox, or other user interface element has a unique name.

Reading CheckBox Data - Servlets

```
<html>
<body>
<h2 align="center"><font color="green">HOBBIES</font></h2>
<form action="Onget" method="post">
<table align="center">
<tr>
<td><font>Playing Cricket</font></td>
<td><input type="checkbox" name="hobbies" value="Playing Cricket"></td>
</tr>
<tr>
<td><font>Watching Movies</font></td>
<td><input type="checkbox" name="hobbies" value="Watching Movies"></td>
</tr>
<tr>
<td><font>Listening Songs</font></td>
<td><input type="checkbox" name="hobbies" value="Listening Songs"></td>
</tr>
<tr>
<td><font>Playing Basketball</font></td>
<td><input type="checkbox" name="hobbies" value="Playing Basketball"></td>
</tr>
<tr>
<td></td>
<td><input type="submit" value="Submit"></td>
</tr>
</table>
</form>
</body>
</html>
```

Reading CheckBox Data - Servlets

```
public class GetParameterValuesExample extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        PrintWriter printWriter=response.getWriter();
        response.setContentType("text/html");
        String[] values=request.getParameterValues("hobbies");
        printWriter.println("Selected values:-");
        for(int i=0; i<values.length; i++){
            printWriter.println("<li>"+values[i]+"</li>");
        }
        printWriter.close();
    }
}
```

Reading Form Data from Servlets

□ **getParameterNames()**

- ▣ Use `getParameterNames` to get the full list of parameter names in the form of an Enumeration.
- ▣ Each entry of this Enumeration can be cast to a `String` and used in a `getParameter` or `getParameterValues` call.
- ▣ If there are no parameters in the current request, `getParameterNames` returns an empty Enumeration (not null).

Reading Form Data from Servlets

□ **getParameterMap()**

- ▣ An alternative to `getParameterNames` is `getParameterMap`.
- ▣ This method returns a `Map`: the parameter names (strings) are the table keys and the parameter values are the table values.
- ▣ Parameter Values are the string arrays as returned by `getParameterNames`.

HttpServlet

- Whenever the web browser fetches a file (a page, a picture, etc) from a web server, it does so using HTTP (Hypertext Transfer Protocol).
- HTTP is a request/response protocol, which means your computer sends a request for some file, and the web server sends back a response.
- HTTP 1.1 defines the following request methods:
- **GET:** Retrieves the resource identified by the request URL
- **HEAD:** Returns the headers identified by the request URL

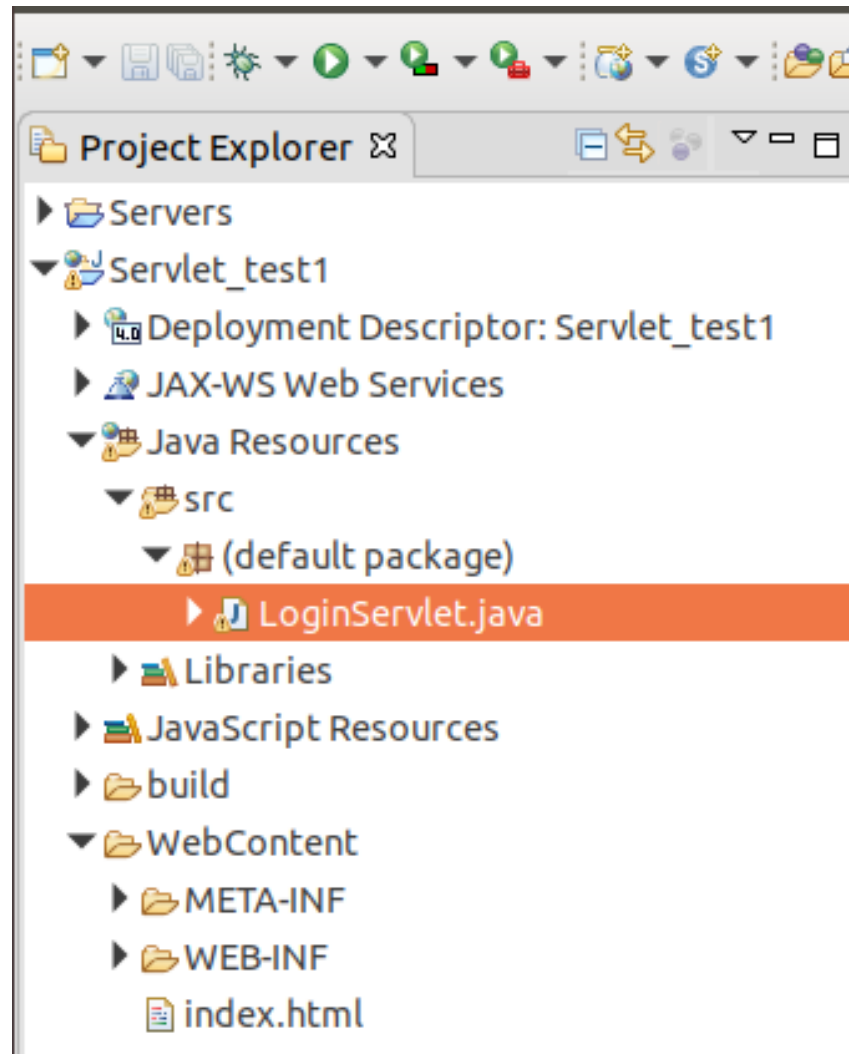
HttpServlet

- ❑ **POST:** Sends data of unlimited length to the Web server
- ❑ **PUT:** Stores a resource under the request URL
- ❑ **DELETE:** Removes the resource identified by the request URL
- ❑ **OPTIONS:** Returns the HTTP methods the server supports
- ❑ **TRACE:** Returns the header fields sent with the TRACE request

HTTP Get request

- **Example**
- **LoginServlet.java:** An HTTP servlet, which is a Java class that extends HttpServlet.
- **index.html:** A web page to invoke the servlet by sending a request. It sends a GET request.

HTTP Get request



HTTP Get request

```
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException{
        PrintWriter pw=res.getWriter();
        res.setContentType("text/html");
        String user=req.getParameter("userName");
        String pass=req.getParameter("userPassword");
        if(user.equals("admin") && pass.equals("admin_1234")) {
            pw.println("Welcome "+ user);
            pw.println("<br>Login Successful...!");
        }
        else
            pw.println("Login Failed...!");
        pw.close();
    }
}
```

HTTP Get request

```
<html>
<head>
</head>
<body>
<form action="LoginServlet" method="get">
  <table>
    <tr>
      <td><font face="verdana" size="2px">Name:</font></td>
      <td><input type="text" name="userName"></td>
    </tr>
    <tr>
      <td><font face="verdana" size="2px">Password:</font></td>
      <td><input type="password" name="userPassword"></td>
    </tr>
  </table>
  <input type="submit" value="Login">
</form>
</body>
</html>
```

HTTP Get request

```
<servlet>
    <servlet-name>GetReqServlet</servlet-name>
    <servlet-class>com.aklc.servlet.GetRequestServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>GetReqServlet</servlet-name>
    <url-pattern>/getReq</url-pattern>
</servlet-mapping>
```

HTTP Get request



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/Servlet_test1/index.html`. The browser has three tabs: `index.html`, `LoginServlet.java`, and `Insert title here`. The page content includes a login form with the following elements:

- A label `Name:` followed by a text input field containing the text `admin`.
- A label `Password:` followed by a password input field containing ten dots.
- A `Login` button located below the password field.

HTTP Get request



HTTP Post request

- **Example**
- **LoginServlet1.java:** An HTTP servlet, which is a Java class that extends HttpServlet.
- **index.html:** A web page to invoke the servlet by sending a request. It sends a GET request.

HTTP Post request

```
@WebServlet("/LoginServlet1")
public class LoginServlet1 extends HttpServlet {
    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
        ServletException, IOException{
        PrintWriter pw=res.getWriter();
        res.setContentType("text/html");
        String user=req.getParameter("userName");
        String pass=req.getParameter("userPassword");
        if(user.equals("admin") && pass.equals("admin_1234")) {
            pw.println("Welcome "+ user);
            pw.println("<br>Login Successful...!");
        }
        else
            pw.println("Login Failed...!");
        pw.close();
    }
}
```

HTTP Post request

```
<html>
<head>
</head>
<body>
<form action="LoginServlet1" method="post">
  <table>
    <tr>
      <td><font face="verdana" size="2px">Name:</font></td>
      <td><input type="text" name="userName"></td>
    </tr>
    <tr>
      <td><font face="verdana" size="2px">Password:</font></td>
      <td><input type="password" name="userPassword"></td>
    </tr>
  </table>
  <input type="submit" value="Login">
</form>
</body>
</html>
```


HTTP Post request



```
<servlet>
    <servlet-name>PostReqServlet</servlet-name>
    <servlet-class>com.aklc.servlet.PostRequestServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>PostReqServlet</servlet-name>
    <url-pattern>/postReq</url-pattern>
</servlet-mapping>
```

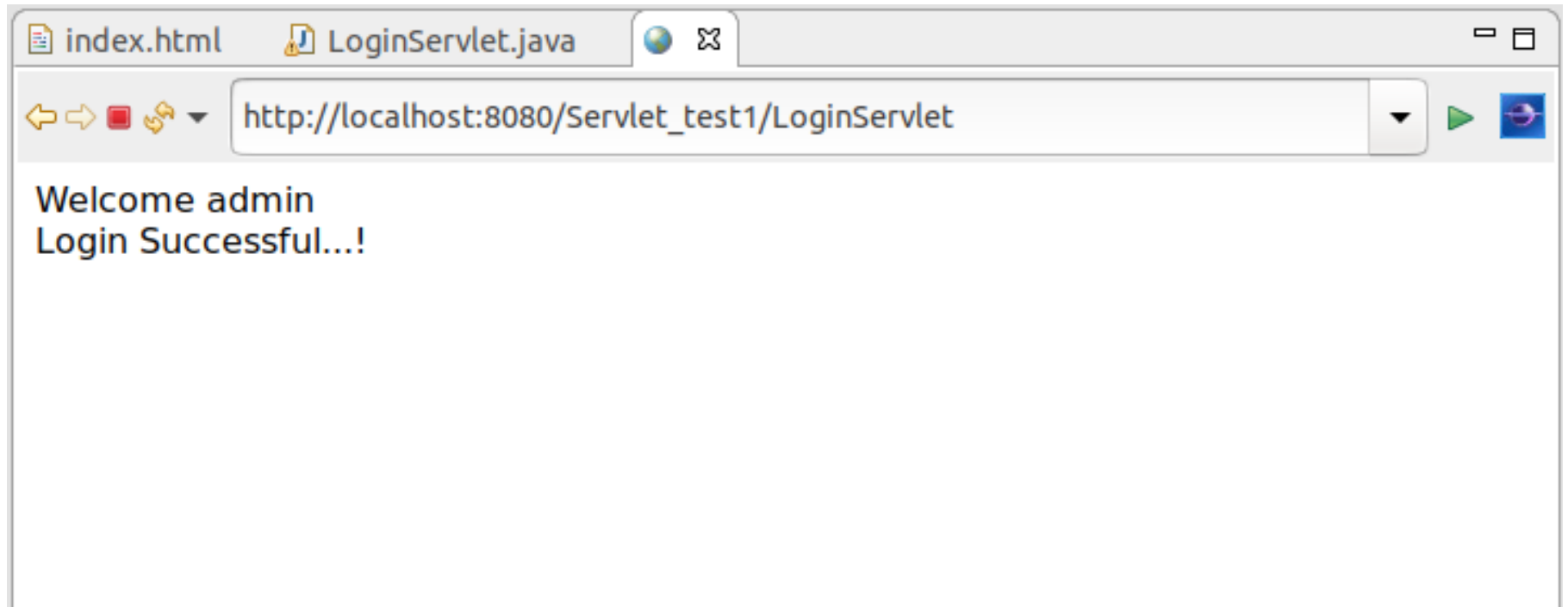
HTTP Post request



The screenshot shows a web browser window with three tabs: 'index.html', 'LoginServlet.java', and 'Insert title here'. The address bar displays the URL 'http://localhost:8080/Servlet_test1/index.html'. The main content area contains a login form with the following elements:

- Name:** A text input field containing the text 'admin'.
- Password:** A text input field with masked characters (dots) and a cursor at the end.
- Login:** A button labeled 'Login'.

HTTP Post request



HTTP Get vs HTTP Post

HTTP Get Request	HTTP Post Request
Get Request sends the request parameter as query string appended at the end of the request.	Post request send the request parameters as part of the http request body.
Example: <code>http://localhost:8080/PractiseApp/getReq?username=Ashok&password=lordshiva</code>	Example: <code>http://localhost:8080/PractiseApp/postReq</code>
Restriction on form data, only ASCII characters allowed.	No Restriction on form data, Binary data is also allowed.
Get methods have maximum size as 2000 character.	Post methods have maximum size is 8 mb.
Restriction on form length, So URL length is restricted	No restriction on form data.
Remain in browser history.	Never remain the browser history.

Cookies

- ❑ Web applications are typically a series of HTTP requests and responses.
- ❑ As HTTP is a stateless protocol, information is not automatically saved between HTTP requests.
- ❑ Web applications use cookies to store state information on the client.
- ❑ Cookies can be used to store information about the user, the user's shopping cart, and so on.
- ❑ Cookies are small bits of textual information that a Web server sends to a browser.

Cookies

- By having the server read cookies it sent the client previously, the site can provide visitors with a number of conveniences like
 - ▣ Identifying a user during e-commerce session
 - ▣ Avoiding username and password entered again and again
 - ▣ Customizing a site based on user's browsing history
 - ▣ Focusing Advertisement

Cookies

- The Cookie class provides an easy way for servlet to read, create, and manipulate HTTP cookies on the web browser.
 - ▣ **getCookies():** To retrieve cookies as request.
 - ▣ **addCookie():** To send a new cookie to the browser.
 - ▣ **setMaxAge():** To set the age of cookie.

Cookies

- Creating a cookie and store in browser
 - ▣ **Cookie ck=new Cookie("key", "value");**
- To add cookie expiry time
 - ▣ **ck.setMaxAge(300); // seconds ie 5 min**
- To add Cookie to HTTP response header.
 - ▣ **response.addCookie(ck);**

Cookies

- To retrieve Cookie from browser
 - ▣ **Cookie cks[]=request.getCookies();**

- To remove a Cookie, just set the value as null and age as 0.
 - ▣ **Cookie ck=new Cookie("key", null);**
 - ▣ **ck.setMaxAge(0);**
 - ▣ **response.addCookie(ck);**

Creating a Cookie

- **Example**
- **WriteCookieServlet.java** : An HTTP servlet, which is a Java class that extends `HttpServlet`.
- **cookieWrite.html** : A web page to invoke the servlet by sending a request.
- **web.xml** : A deployment descriptor which will map the request from the web page to the servlet.

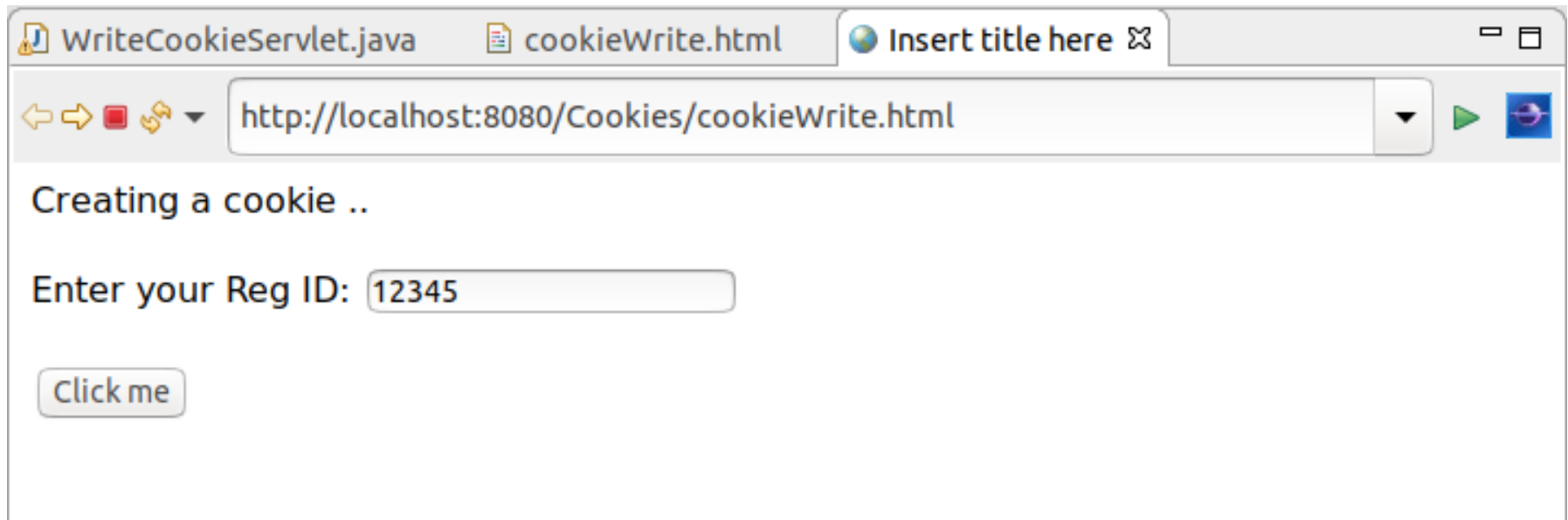
Creating a Cookie

```
<html>
  <body>
    Creating a cookie .. <br/><br/>
    <form action='WriteCookieServlet' >
      Enter your Reg ID:
      <input type='text' name='id' />
      <br/><br/>
      <input type='submit' value='Click me' />
    </form>
  </body>
</html>
```

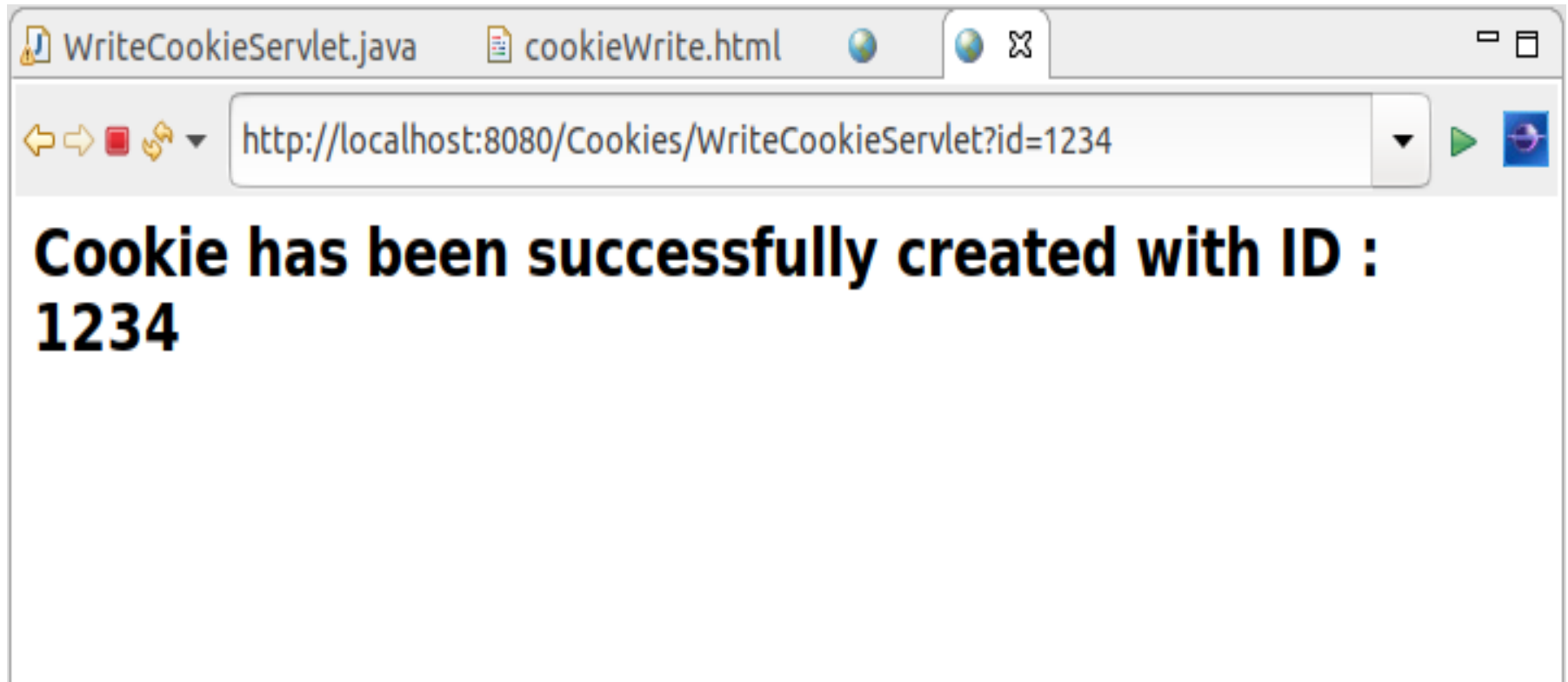
Creating a Cookie

```
@WebServlet("/WriteCookieServlet")
public class WriteCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        String id = req.getParameter("id");
        Cookie mycookie = new Cookie("RegID", id);
        resp.addCookie(mycookie);
        resp.setContentType("text/html");
        PrintWriter pw = resp.getWriter();
        pw.println("<h2> Cookie has been successfully created .. </h2>");
        pw.close();
    }
}
```

Creating a Cookie



Creating a Cookie



Reading a Cookie

- **Example**
- **ReadCookieServlet.java** : An HTTP servlet, which is a Java class that extends HttpServlet.
- **cookieRead.html** : A web page to invoke the servlet by sending a request.
- **web.xml** : A deployment descriptor which will map the request from the web page to the servlet.

Reading a Cookie

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

public class ReadCookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        Cookie[] cookies = req.getCookies();
        String name="", value="";
        for (int i=0;i<cookies.length;i++) {
            name = cookies[i].getName();
            if (name.equals("RegID")) {
                value=cookies[i].getValue();
                break;
            }
        }
        resp.setContentType("text/html");
        PrintWriter pw = resp.getWriter();
        pw.println("<h2> Welcome, Stored RegID is .. "+value+"</h2>");
        pw.close();
    }
}
```


Reading a Cookie

```
<html>
```

```
  <body>
```

```
    Reading a cookie .. <br/><br/>
```

```
    <form action='readCookie' >
```

```
      <input type='submit' value='Click me' />
```

```
    </form>
```

```
  </body>
```

```
</html>
```

Reading a Cookie

```
<servlet>
    <servlet-name>readCookie</servlet-name>
    <servlet-class>com.aklc.servlet.ReadCookieServlet</servlet-class>
</servlet>

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

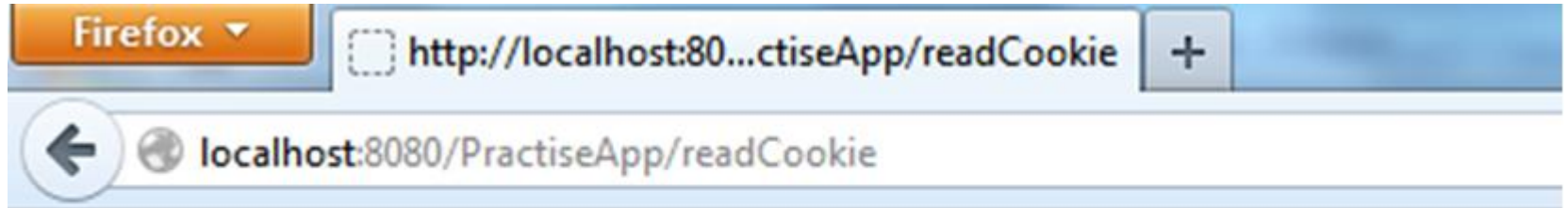
Reading a Cookie



Reading a cookie ..

Click me

Reading a Cookie



Welcome, Stored RegID is .. 1234

Session Handling

- HTTP is a “stateless” protocol.
- Each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server does not automatically maintain contextual information about the client.
- No built-in support for maintaining contextual information in Server.
- There are 4 typical solutions to this problem:
 - ▣ Cookies
 - ▣ URL rewriting
 - ▣ Hidden form fields
 - ▣ HttpSession

Session Handling

□ Cookies:

- A web server can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

□ URL rewriting:

- You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.
- `http://jmaster.in/home.html;sessionid=12345`

Session Handling

❑ 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 request.
- ❑ Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

Session Handling

□ HttpSession:

- 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.

Session Tracking Basics

- Using sessions in servlets is straightforward and involves four basic steps.
- **Accessing the session object associated with the current request**
 - ▣ Call `request.getSession()` to get an `HttpSession` object, which is a simple hash table for storing user-specific data.
- **Looking up information associated with a session**
 - ▣ Call `getAttribute()` on the `HttpSession` object, cast the return value to the appropriate type, and check whether the result is null.

Session Tracking Basics

- **Storing information in a session**
 - ▣ Use `setAttribute()` with a key and a value.

- **Discarding session data**
 - ▣ Call `removeAttribute()` to discard a specific value. Call `invalidate()` to discard an entire session.
 - ▣ Call `logout()` to log the client out of the Web server and invalidate all session associated with that user.

Session Handling Example - 1

- ❑ **Example**
- ❑ **Validate.java** : An HTTP servlet, which is a Java class that extends HttpServlet. It checks for Login name & password and if valid user then create a new session. Redirect to new page and display details.
- ❑ **Welcome.java** : An HTTP servlet, which is a Java class that extends HttpServlet. This will display the user attribute using session object.
- ❑ **index.html** : A web page to invoke the Validate servlet by sending a username & password request.

Session Handling Example - 1

```
//index.html
```

```
<form method="post" action="Validate">  
  User: <input type="text" name="user" /><br/>  
  Password: <input type="text" name="pass" ><br/>  
  <input type="submit" value="submit">  
</form>
```

Session Handling Example - 1

```
//Validate.java
public class Validate extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            //creating a session
            HttpSession session = request.getSession();
            session.setAttribute("user", name);
            response.sendRedirect("Welcome");
        }
    }
}
```

Session Handling Example - 1

```
//welcome.java
public class Welcome extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

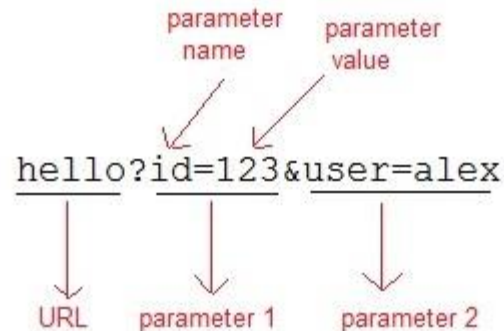
        HttpSession session = request.getSession();

        String user = (String)session.getAttribute("user");
        out.println("Hello "+user);
    }
}
```

Session Handling Example - 2

- ❑ **Example using URL rewriting**
- ❑ **MyServlet.java** : An HTTP servlet, which is a Java class that extends HttpServlet. It checks for Login name & password and if valid user then redirect to new page. Rewrite the URL to send the user details.
- ❑ **First.java** : An HTTP servlet, which is a Java class that extends HttpServlet. This will display the user attribute using request object.
- ❑ **index.html** : A web page to invoke the servlet by sending a username & password request.

Session Handling Example - 2



```
//index.html
```

```
<form method="post" action="validate">  
  Name:<input type="text" name="user" /><br/>  
  Password:<input type="text" name="pass" ><br/>  
  <input type="submit" value="submit">  
</form>
```


Session Handling Example - 2

//MyServlet.java

```
public class MyServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        String name = request.getParameter("user");
        String pass = request.getParameter("pass");

        if(pass.equals("1234"))
        {
            response.sendRedirect("First?user_name="+ name);
        }
    }
}
```

Session Handling Example - 2

//First.java

```
public class First extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        String user = request.getParameter("user_name");  
        out.println("Welcome "+user);  
    }  
}
```

Session Handling Example - 3

□ Example using Hidden Fields

```
//index.html
```

```
<form method="post" action="validate">  
  Name:<input type="text" name="user" /><br/>  
  Password:<input type="text" name="pass" ><br/>  
  <input type="submit" value="submit">  
</form>
```

Session Handling Example - 3

```
//First.java
```

```
public class First extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
                           response) throws ServletException, IOException {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        //getting value submitted in form from HTML file
        String user = request.getParameter("user");

        //creating a new hidden form field
        out.println("<form action='Second'>");
        out.println("<input type='hidden' name='user' value='"+user+"'>");
        out.println("<input type='submit' value='submit' >");
        out.println("</form>");
    }
}
```

Session Handling Example - 3

```
//Second.java
```

```
public class Second extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
                           response) throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
  
        //getting parameter from the hidden field  
        String user = request.getParameter("user");  
        out.println("welcome "+user);  
    }  
}
```

Session Handling Example - 4

- **Example**
- **SessionTrackServlet.java** : An HTTP servlet, which is a Java class that extends HttpServlet.
- **sessionTrack.html** : A web page to invoke the servlet by sending a request.
- **web.xml** : A deployment descriptor which will map the request from the web page to the servlet.

Session Handling Example - 4

```
public class SessionTrackServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        HttpSession session = req.getSession(true);
        Date createTime = new Date(session.getCreationTime());
        Date lastAccessTime = new Date(session.getLastAccessedTime());
        String title = "Welcome Back to my website";
        int visitCount = 0;
        String userID="";
        // Check if this is new comer on your web page.
        if (session.isNew()){
            title = "Welcome to my website";
            session.setAttribute("userID", "ABCD");
        }
    }
}
```

Session Handling Example - 4

```
else
{
    visitCount = (Integer)session.getAttribute("visitCount");
    visitCount = visitCount + 1;
    userID = (String)session.getAttribute("userID");
}
session.setAttribute("visitCount", visitCount);
resp.setContentType("text/html");
PrintWriter pw = resp.getWriter();
pw.println("<h1>Session Infomation</h1>" + "<br/>" +
    "ID : " + session.getId() + "<br/>" +
    "Creation Time : " + createTime + "<br/>" +
    "Time of Last Access : " + lastAccessTime + "<br/>" +
    "User ID : " + userID + "<br/>" +
    "Number of visits : " + visitCount + "<br/>" +
}
```


Session Handling Example - 4

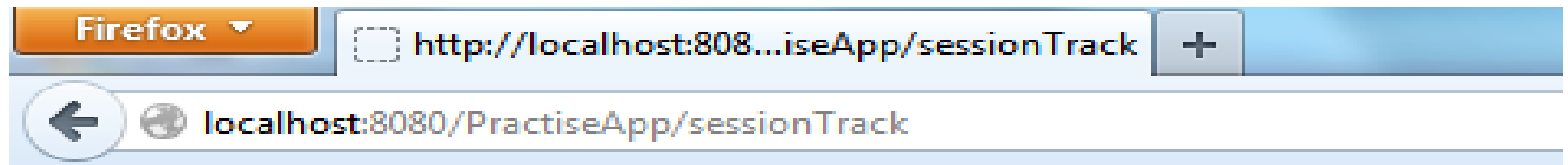
```
//sessionTrack.html
```

```
<html>
    <body>
        Session Handling DEMO <br/><br/>
        <a href='sessionTrack'> Visit my website </a>
    </body>
</html>
```

```
//web.xml
```

```
<servlet>
    <servlet-name>SessionTrack</servlet-name>
    <servlet-class>com.aklc.servlet.SessionTrackServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>SessionTrack</servlet-name>
    <url-pattern>/sessionTrack</url-pattern>
</servlet-mapping>
```

Session Handling Example - 4

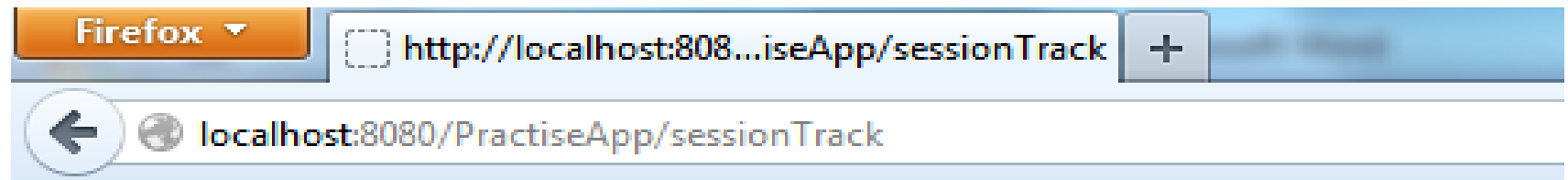


Welcome to my website

Session Information

Session info	value
id	3A55D7FCE4BEFD04D95DB043308266C8
Creation Time	Sat Jul 20 14:31:40 IST 2013
Time of Last Access	Sat Jul 20 14:31:40 IST 2013
User ID	
Number of visits	0

Session Handling Example - 4



Welcome Back to my website

Session Information

Session info	value
id	3A55D7FCE4BEFD04D95DB043308266C8
Creation Time	Sat Jul 20 14:31:40 IST 2013
Time of Last Access	Sat Jul 20 14:31:40 IST 2013
User ID	ABCD
Number of visits	1

Servlet Collaboration

- The exchange of information among servlets of a particular Java web application is known as Servlet Collaboration.
- This enables passing/sharing information from one servlet to the other through method invocations.
- The servlet api provides two interfaces namely:
 - ▣ `javax.servlet.RequestDispatcher`
 - ▣ `javax.servlet.http.HttpServletResponse`

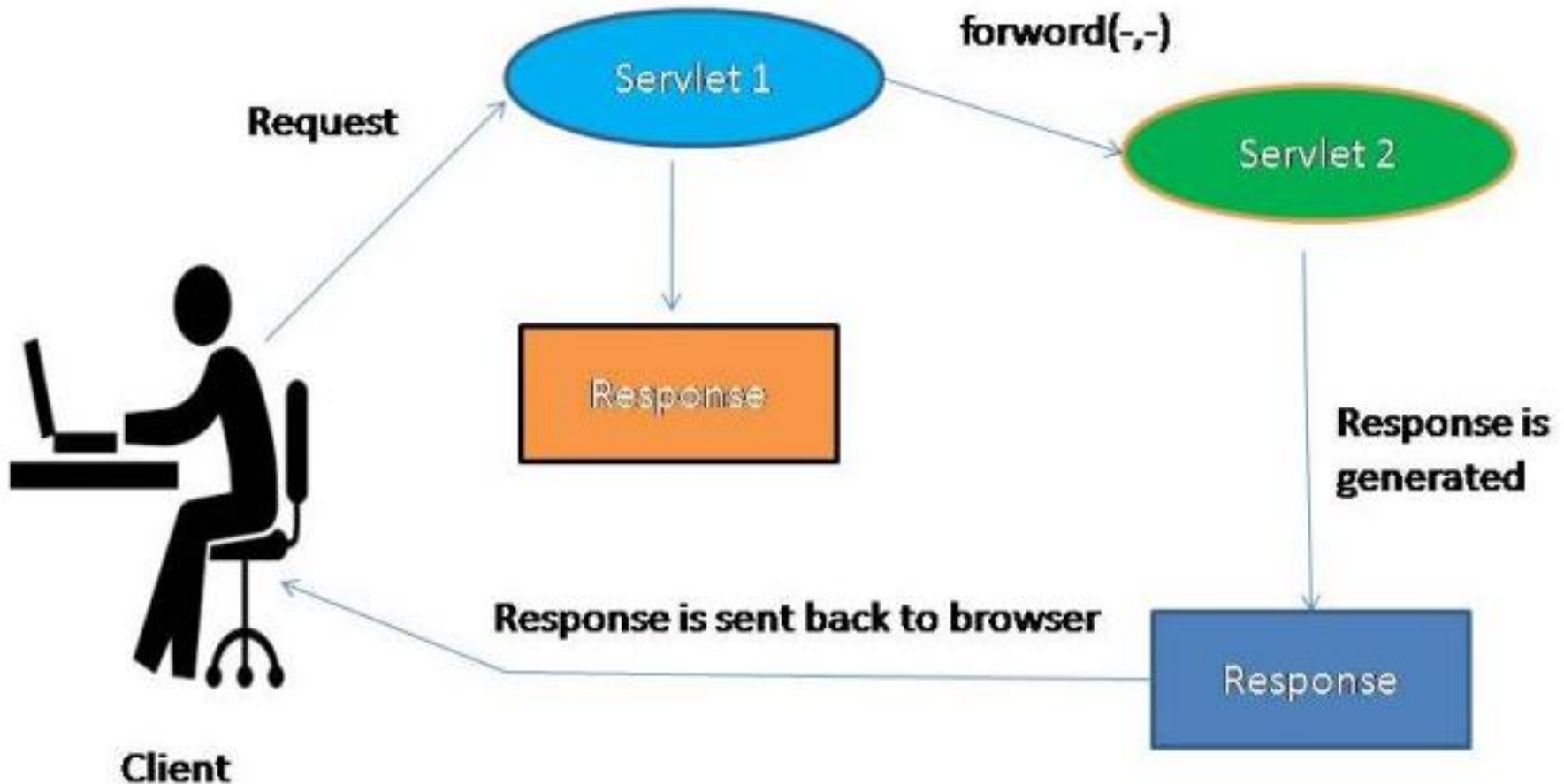
RequestDispatcher in Servlet

- ❑ The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- ❑ This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.
- ❑ There are two methods defined in the RequestDispatcher interface.
 - ❑ **forward()**
 - ❑ **include()**

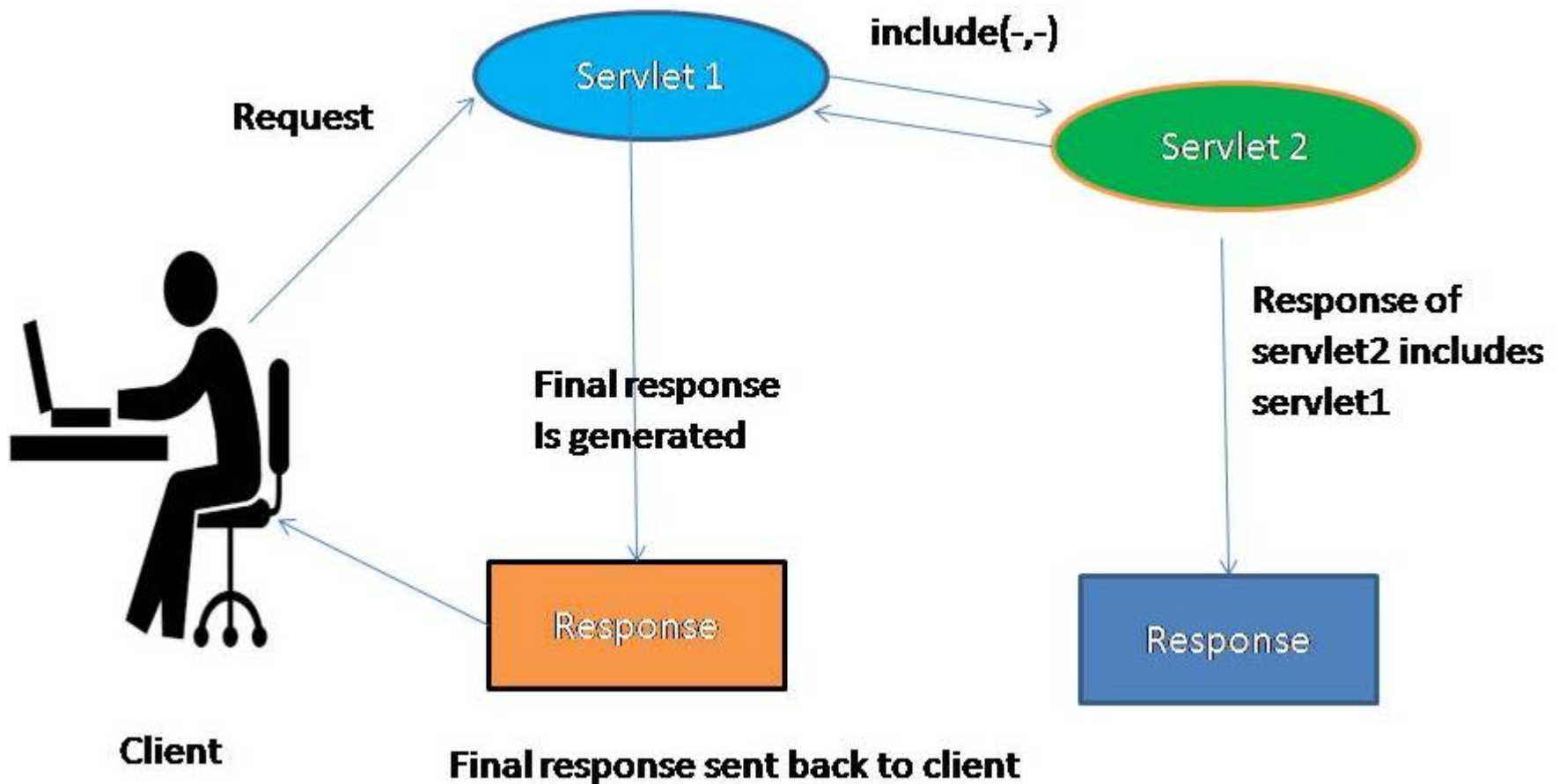
RequestDispatcher in Servlet

- ❑ **public void forward** (ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:
 - Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- ❑ **public void include** (ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:
 - Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

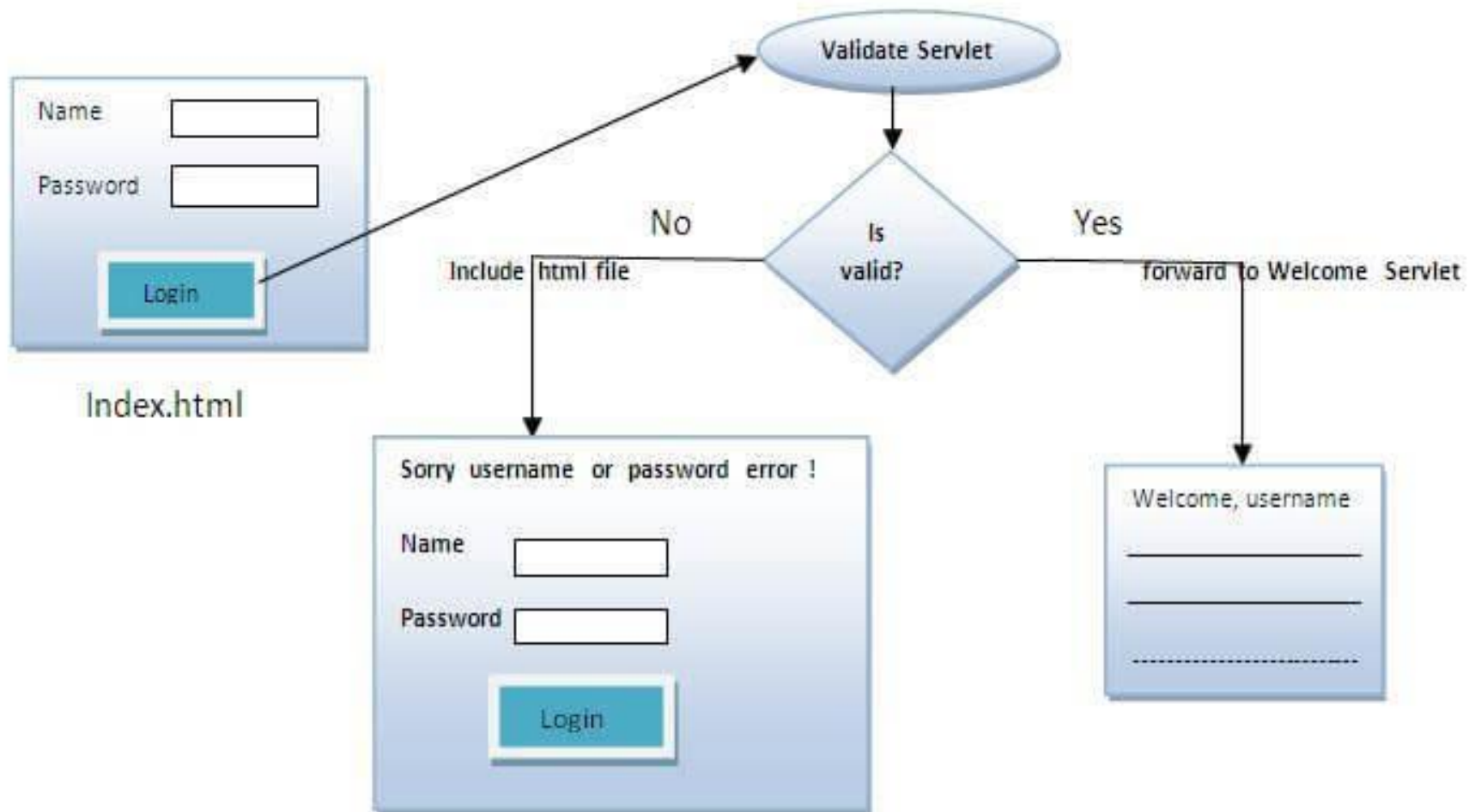
forward() in RequestDispatcher



include() in RequestDispatcher



Example on RequestDispatcher



Example on RequestDispatcher

```
<html>
  <body>
    <form action="servlet1" method="post">
      Name:<input type="text" name="userName"/><br/>
      Password:<input type="password" name="userPass"/><br/>
      <input type="submit" value="login"/>
    </form>
  </body>
</html>
```

Example on RequestDispatcher

```
public class Login extends HttpServlet {  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        String n=request.getParameter("userName");  
        String p=request.getParameter("userPass");  
        if(p.equals("servlet"){  
            RequestDispatcher rd=request.getRequestDispatcher("servlet2");  
            rd.forward(request, response);  
        }  
        else{  
            out.print("Sorry UserName or Password Error!");  
            RequestDispatcher rd=request.getRequestDispatcher("/index.html");  
            rd.include(request, response);  
        }  
    }  
}
```

Example on RequestDispatcher

```
//WelcomeServlet.java
```

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

```
public class WelcomeServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String n=request.getParameter("userName");
        out.print("Welcome "+n);
    }
}
```

Example on RequestDispatcher



localhost:8888/dispatcher/

Name:

Password:



localhost:8888/dispatcher/go?userName=k&userPass=k

Sorry username or password error!

Name:

Password:



localhost:8888/dispatcher/go?userNas=k

Sorry username or password error!

Name:

Password:



localhost:8888/dispatcher/go?userName=sonoo&servlet

Welcome sonoo

sendRedirect() of HttpServletResponse

- ❑ The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- ❑ It accepts relative as well as absolute URL.
- ❑ It works at client side because it uses the url bar of the browser to make another request.
- ❑ It can work inside and outside the server.

forward() vs sendRedirect()

forward()	sendRedirect()
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
<code>request.getRequestDispatcher("servlet2").forward(request, response);</code>	<code>response.sendRedirect("servlet2");</code>

Example on sendRedirect()

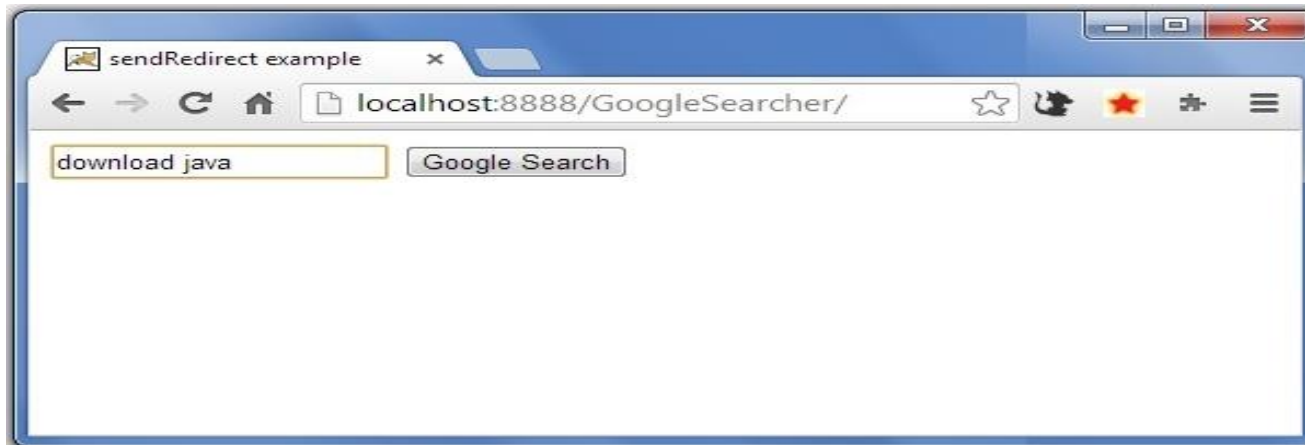
```
<html>  
  <body>  
    <form action="MySearcher">  
      <input type="text" name="name">  
      <input type="submit" value="Google Search">  
    </form>  
  </body>  
</html>
```


Example on sendRedirect()

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MySearcher extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        String name=request.getParameter("name");
        response.sendRedirect("https://www.google.co.in/#q="+name);
    }
}
```

Example on sendRedirect()



Accessing the Web Context

- The context in which web components execute is an object that implements the `ServletContext` interface.
- An object of `ServletContext` is created by the web container at time of deploying the project.
- You retrieve the web context by using the `getServletContext` method.
- The web context provides methods for accessing following data from `web.xml`.
 - ▣ Initialization parameters
 - ▣ Resources associated with the web context
 - ▣ Object-valued attributes
 - ▣ Logging capabilities

Accessing the Web Context

- Advantage of ServletContext
 - ▣ Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet.
 - ▣ We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet.
 - ▣ Thus it removes maintenance problem.
- Usage of ServletContext Interface
 - ▣ The object acts as an interface between the container and servlet.
 - ▣ To get configuration information from the web.xml file.
 - ▣ To set, get or remove attribute from the web.xml file.
 - ▣ To provide inter-application communication.

Servlet Init Parameters

- You can pass parameters to a servlet from the web.xml file.
- The init parameters of a servlet can only be accessed by that servlet.
- Here is how you configure them in the web.xml file:

```
<servlet>
  <init-param>
    <param-name>myParam</param-name>
    <param-value>paramValue</param-value>
  </init-param>
</servlet>
```

Example on Accessing Web Context

```
public class DemoServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

        //creating ServletContext object
        ServletContext context=getServletContext();

        //Getting the value of the initialization parameter and printing it
        String driverName=context.getInitParameter("dname");
        pw.println("driver name is="+driverName);

        pw.close();
    }
}
```

Example on Accessing Web Context

```
<web-app>
```

```
    <servlet>
```

```
        <servlet-name>DemoServlet</servlet-name>
```

```
        <servlet-class>DemoServlet</servlet-class>
```

```
    </servlet>
```

```
    <context-param>
```

```
        <param-name>dname</param-name>
```

```
        <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
```

```
    </context-param>
```

```
    <servlet-mapping>
```

```
        <servlet-name>DemoServlet</servlet-name>
```

```
        <url-pattern>/context</url-pattern>
```

```
    </servlet-mapping>
```

```
</web-app>
```

THANK YOU

