# Java & JEE Training

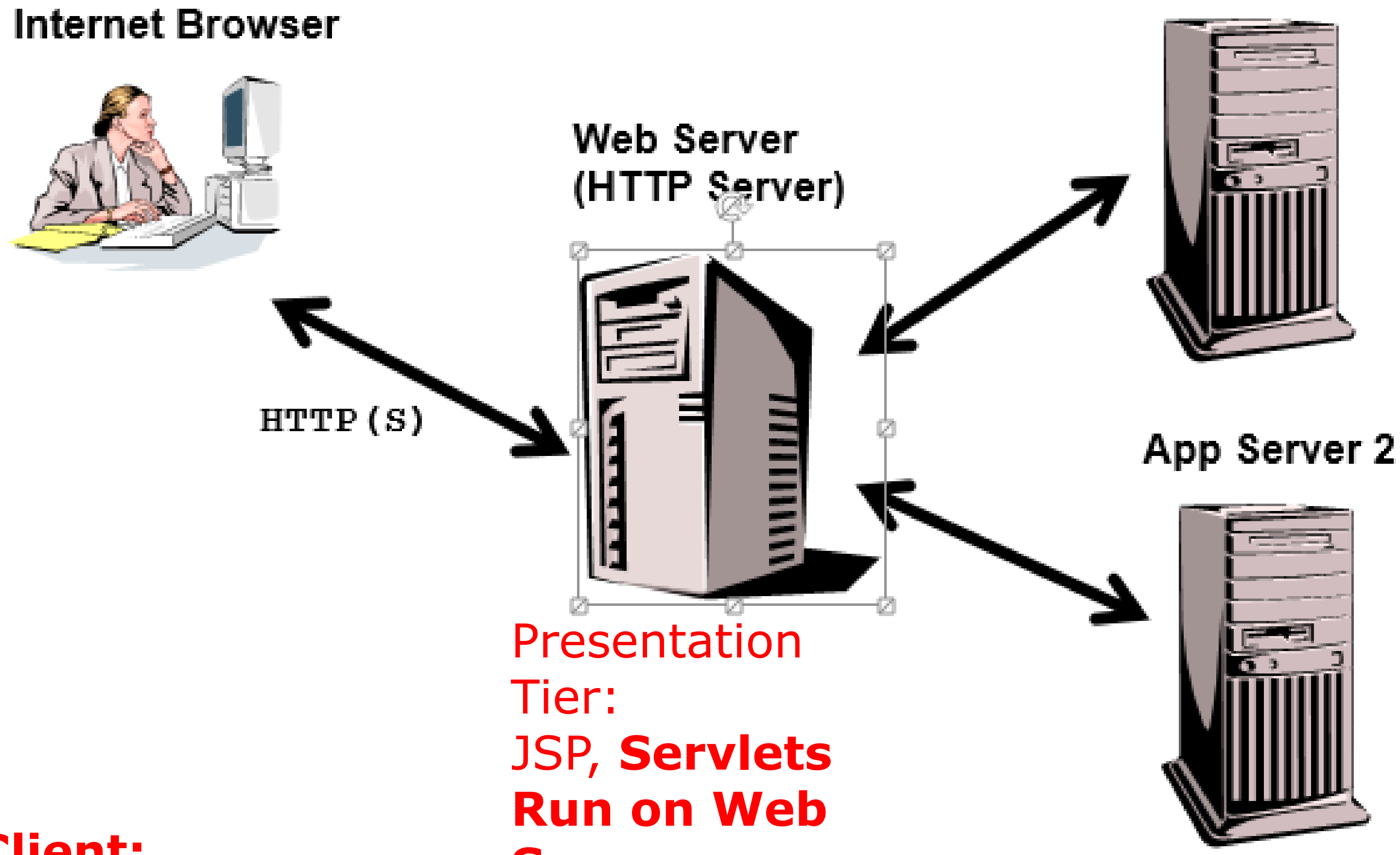
## Day 31 – Session Management using Servlets +Design Patterns

MindsMapped Consulting

## Review of previous session…

….

## Web Server and Application Server



Presentation Tier:
JSP, **Servlets Run on Web Server or container**

**JDBC on web server or app server**

**Client:
HTML JS CSS
(Run on client machine on the browser)**

**Database**

# Java & JEE Training

## Understanding scope: parameters, attributes

MindsMapped Consulting

## Web.xml file: Servlet mapping to URL

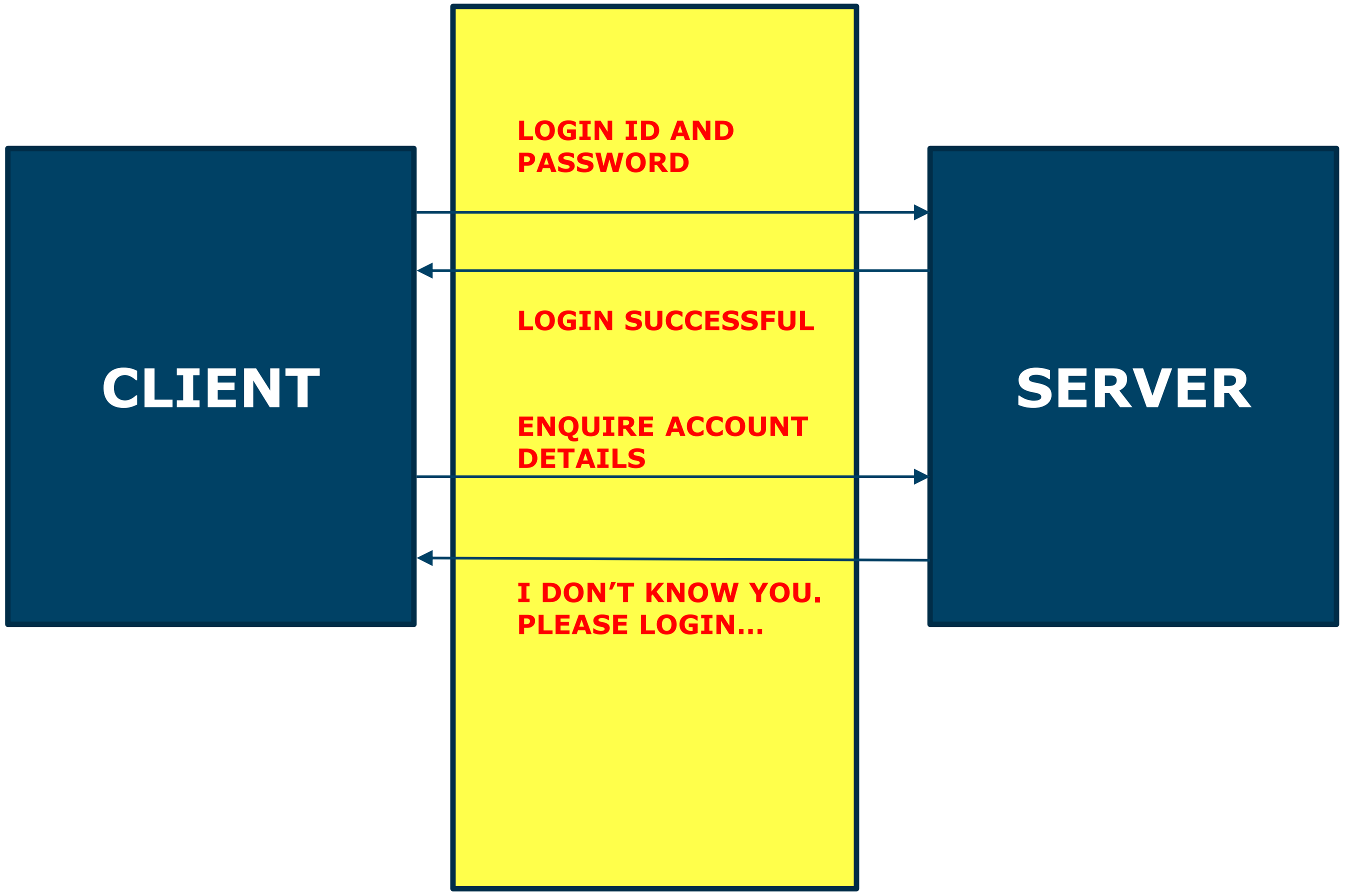
```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <display-name>HelloWorld Application</display-name>
    <description>
        This is a simple web application with a source code organization
        based on the recommendations of the Application Developer's Guide.
    </description>

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>examples.Hello</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

## Requests, Sessions, Application

- **Request**: The client sending an HTTP request to the server.
- **Session**: Is a set of requests coming in from the client from the point that the client logs in to the point that the client logs out.
- **Application**: The time that the application (or server) is up and running.

- An application may have multiple sessions.
- A session may have multiple requests.

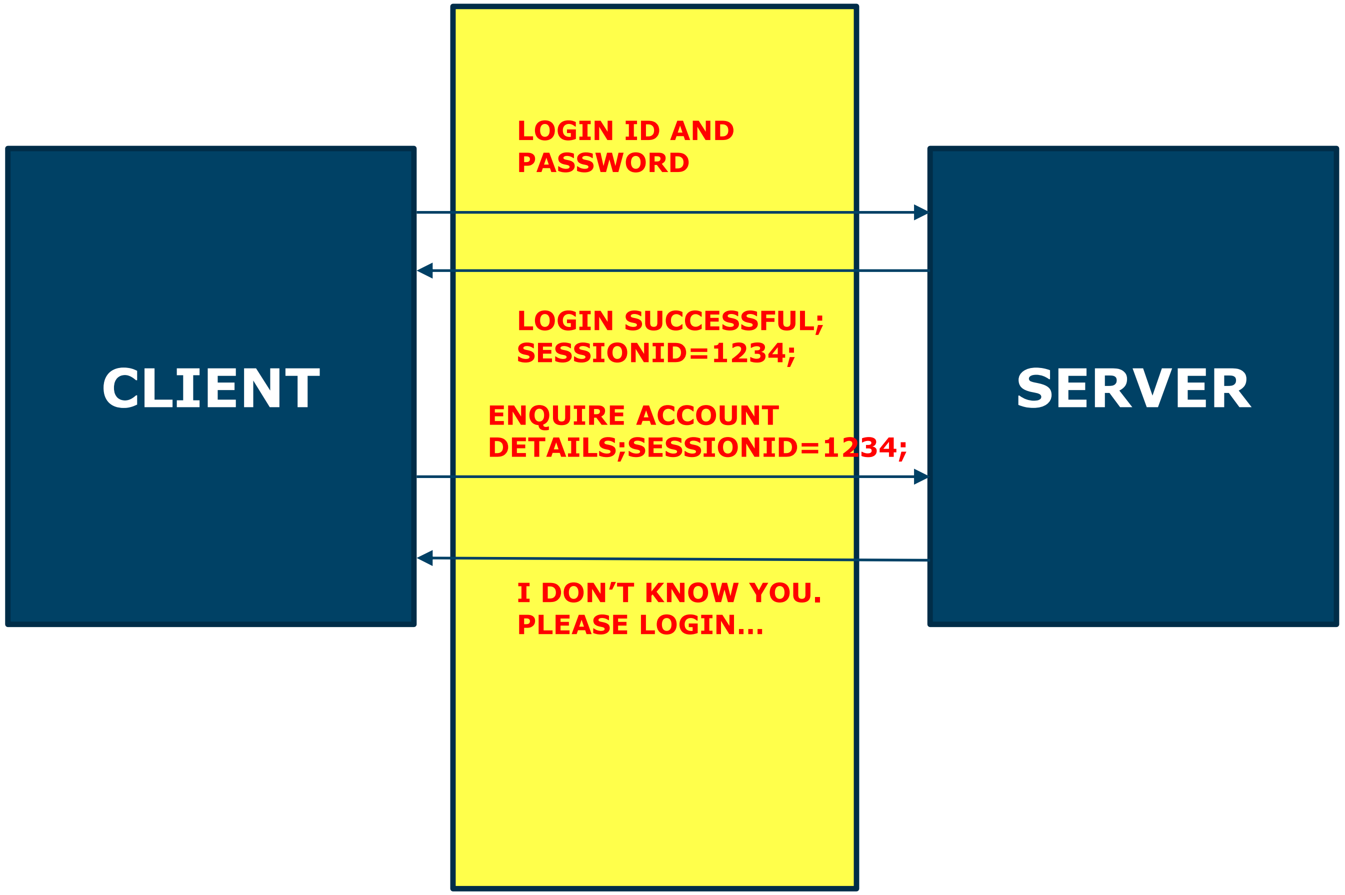
## HTTP IS STATELESS



**HOW DO I MAINTAIN SESSIONS?**

**SESSION**
**- COOKIES…**

## HTTP IS STATELESS


LOGIN ID AND PASSWORD
LOGIN SUCCESSFUL; SESSIONID=1234;
CLIENT
SERVER
ENQUIRE ACCOUNT DETAILS;SESSIONID=1234;
I DON'T KNOW YOU. PLEASE LOGIN…

**SESSION**
**- COOKIES**

**HOW DO I MAINTAIN SESSIONS?**

## Parameters vs Attributes

- Parameters may come into our application **from the client request, or may be configured through deployment descriptor (web.xml)** elements or their corresponding annotations. When you submit a form, form values are sent as request parameters to a web application. In case of a GET request, these parameters are exposed in the URL as name value pairs and in case of POST, parameters are sent within the body of the request.
- Servlet init parameters and context init parameters are set through the deployment descriptor (web.xml) or their corresponding annotations. All parameters are **read-only** from the application code. We have methods in the Servlet API to retrieve various parameters.
- **Attributes** are objects that are attached to various scopes and can be modified, retrieved or removed. Attributes can be read, created, updated and deleted by the web container as well as our application code. We have methods in the Servlet API to add, modify, retrieve and remove attributes. When an object is added to an attribute in any scope, it is called **binding** as the object is bound into a scoped attribute with a given name.

## Parameters vs Attributes

- Parameters are read only Strings, attributes are read/write objects.
- Parameters are String objects, attributes can be objects of any type.

## Methods to manipulate parameters

AVAILABLE FOR THE REQUEST

- **Request**: (Request time constant)

  ServletRequest.**getParameter**(paramname);

  ServletRequest.**getParameterNames**();

AVAILABLE FOR THE SERVLET

- **ServletConfig** init parameters: (Deployment time constant)

  ServletConfig.**getInitParameterNames**()

  ServletConfig.**getInitParameter**(String paramName)

- **ServletContext** init parameters: (Deployment time constant)

  ServletContext.**getInitParameterNames**()

  ServletContext.**getInitParameter**(String paramName)

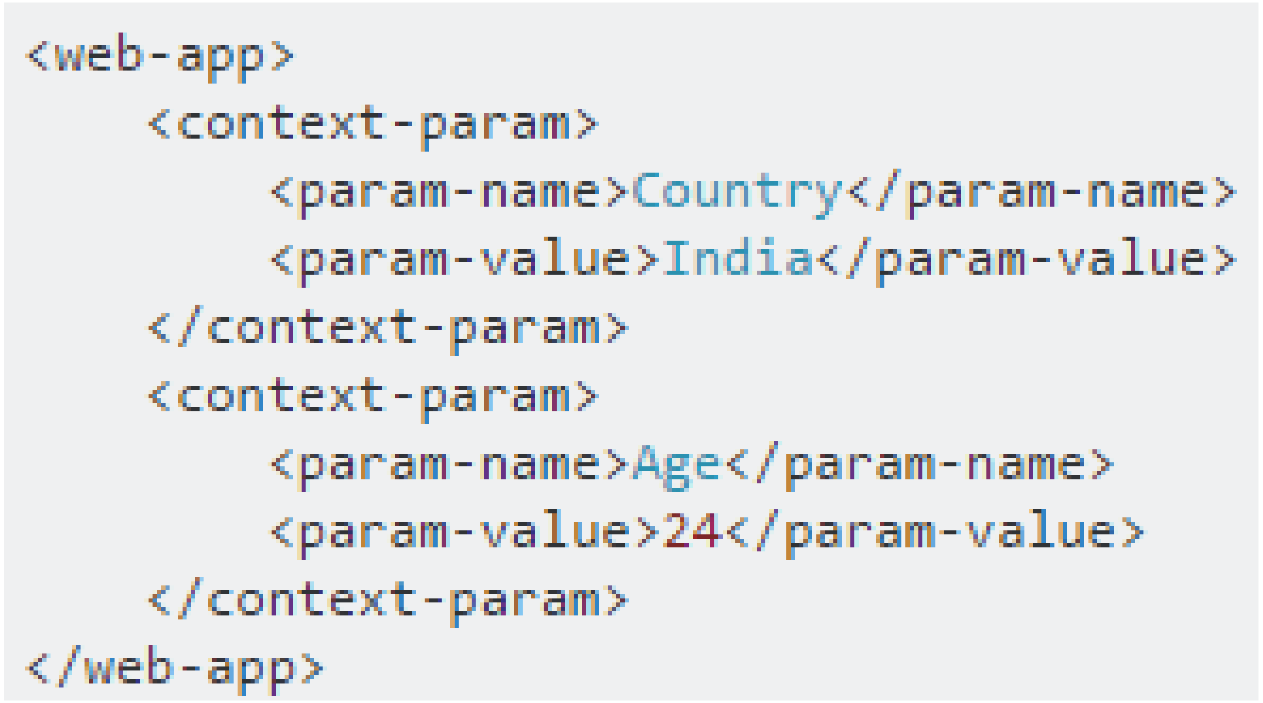
AVAILABLE FOR THE WHOLE APPLICATION

## Web.xml : init parameters (Servlet Scope)

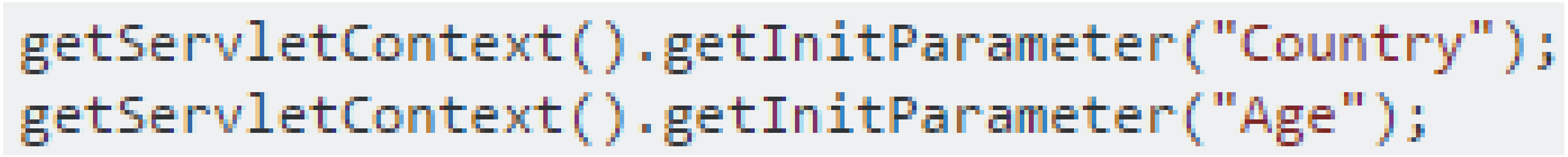
```xml
<servlet>
     <display-name>HelloWorldServlet</display-name>
     <servlet-name>HelloWorldServlet</servlet-name>
     <init-param>
         <param-name>Greetings</param-name>
         <param-value>Hello</param-value>
     </init-param>
</servlet>
```

```java
out.println(getInitParameter("Greetings"));
```

## Web.xml: Init parameters (Application scope)



```xml
<web-app>
    <context-param>
        <param-name>Country</param-name>
        <param-value>India</param-value>
    </context-param>
    <context-param>
        <param-name>Age</param-name>
        <param-value>24</param-value>
    </context-param>
</web-app>
```



```java
getServletContext().getInitParameter("Country");
getServletContext().getInitParameter("Age");
```

## ServletConfig vs ServletContext

- **ServletConfig** **// Available for a specific servlet**
  - ServletConfig available in javax.servlet.*; package
  - ServletConfig object is one per servlet class
  - Object of ServletConfig will be created during initialization process of the servlet
  - This Config object is public to a particular servlet only
  - *Scope*: As long as a servlet is executing, ServletConfig object will be available, it will be destroyed once the servlet execution is completed.
  - We should give request explicitly, in order to create ServletConfig object for the first time
  - In web.xml – <*init-param*> tag will be appear under <*servlet-class*> tag

- **ServletContext** **// Available for whole application**
  - ServletContext available in javax.servlet.*; package
  - ServletContext object is global to entire web application
  - Object of ServletContext will be created at the time of web application deployment
  - *Scope*: As long as web application is executing, ServletContext object will be available, and it will be destroyed once the application is removed from the server.
  - ServletContext object will be available even before giving the first request
  - In web.xml – <*context-param*> tag will be appear under <*web-app*> tag

## How to get servletContext?

```
getServletConfig( ).getServletContext( );
request.getSession( ).getServletContext( );
getServletContext( );
request.getServletContext();
```

## Understanding scope of attributes

- Similar to scope and lifetime of variables in Java as you have seen in blocks and methods in java, parameters and attributes in a Java EE web application also have scope and lifetime in the context of the web application.
- The scope of a parameter/attribute denotes the availability of that parameter/attribute for use. A web application serves multiple requests from clients when it is up and running. These requests can be from same client or different clients. We have seen from the servlet life cycle that a servlet’s service() method is called every time a request comes.

## Scoping in Servlets and JSP

- Request scope
- Session scope
- Application or context scope
- Page scope (only for JSP)

## Request Scope

- Request scope start from the moment an HTTP request hits a servlet in our web container and end when the servlet is done with delivering the HTTP response.
- With respect to the servlet life cycle, the request scope begins on entry to a servlet’s service() method and ends on the exit from that method.
- A ‘request’ scope parameter/attribute can be accessed from any of servlets or jsps that are part of serving one request. For example, you call one servlet/jsp, it then calls another servlet/jsp and so on, and finally the response is sent back to the client.
- **Request** scope is denoted by javax.servlet.http.**HttpServletRequest** interface.
- Container passes the request object as an argument of type HttpServletRequest to Servlet's service method.
- Request object is available in a JSP page as an **implicit object** called **request**. You can set value for an attribute in request object from a servlet and get it from a JSP within the same request using the implicit request object.

## Session scope

- A session scope starts when a client (e.g. browser window) establishes connection with our web application till the point where the browser window is closed.
- Session scope spans across multiple requests from the same client.
- A notable feature of tabbed browsing is that session is shared between the tabs and hence you can requests from other tabs too during a session without logging in again. For instance, you can load your Gmail inbox in another tab without logging in again. This also means browsing an unknown site and a secure site from different tabs from the same browser can expose your secure session ID to malicious applications. So always open a new browser window when you want to do secure transactions, especially financial transactions.
- **Session** scope is denoted by javax.servlet.http.**HttpSession** interface.
- Session object is available in a JSP page as an **implicit object called session**.
- In a servlet, you can get Session object by calling **request.getSession()**.

## Application or context scope

- Context scope or application scope starts from the point where a web application is put into service (started) till it is removed from service (shutdown) or the web application is reloaded. Parameters/attributes within the application scope will be available to all requests and sessions.
- **Application** scope is denoted by javax.servlet.**ServletContext** interface.
- Application object is available in a JSP page as an **implicit object** called **application**.
- In a servlet, you can get application object by calling **getServletContext()** from within the servlets code directly (the servlet itself implements the ServletConfig interface that contains this method) or by explicitly calling **getServletConfig().getServletContext()**.
- The web container provides one ServletContext object per web application per JVM.

## Page scope (Only for JSP, not for Servlets)

- The page scope restricts the scpoe and lifetime of attributes to the same page where it was created.
- **Page** scope is denoted by javax.servlet.jsp.**PageContext** abstract class.
- It is available in a JSP page as an **implicit object** called **pageScope**

## Servlets – Scope at a glance

- Application scope

```
request.getServletContext();
request.getServletContext().setAttribute("attribute_name","value")
```

- Session scope

```
request.getSession(); //going to create the session if session is not
exist.
request.getSession(false); // Not going to create the session.
session.getAttribute("attribute_name");
```

- Request scope

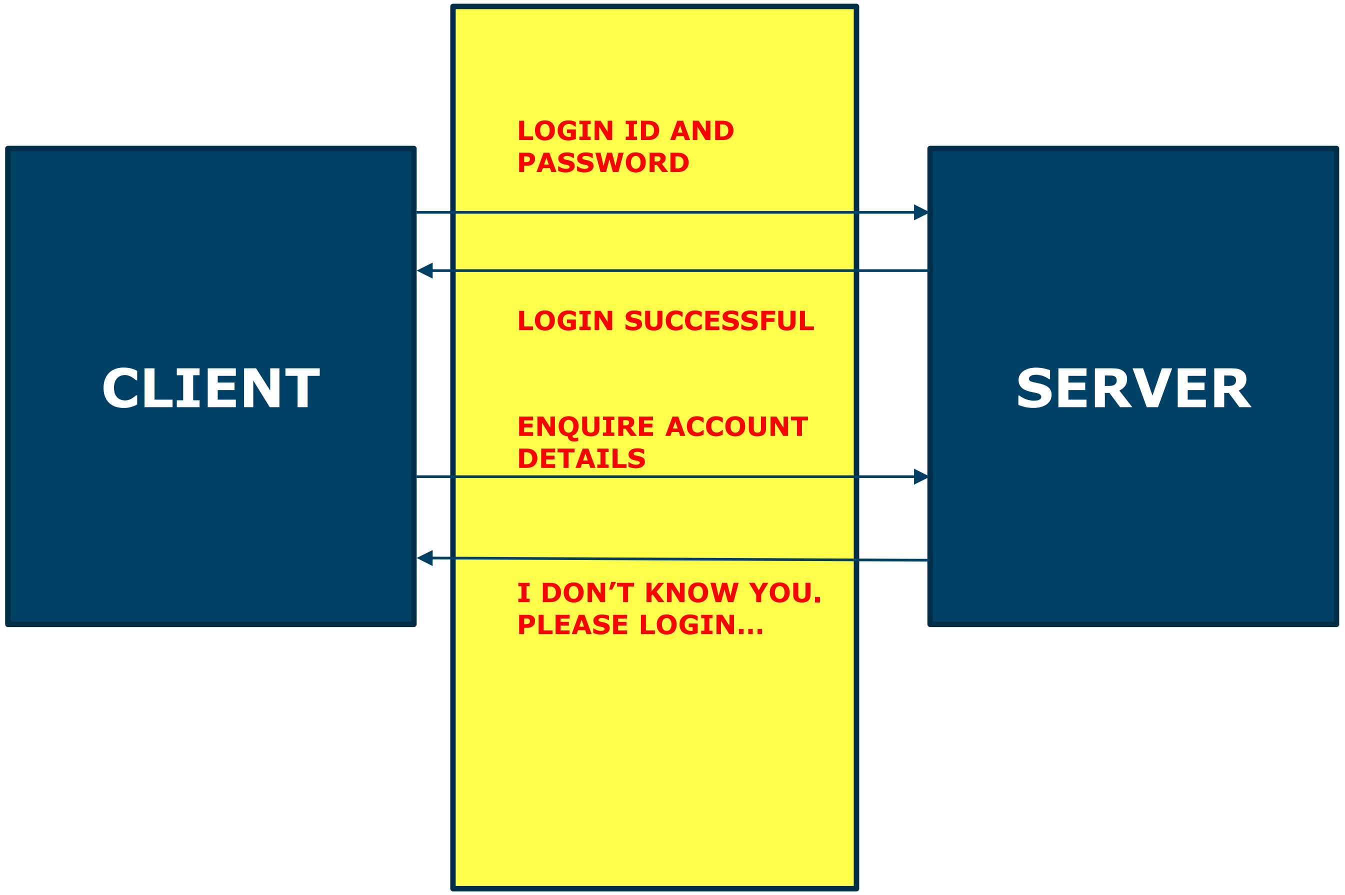
```
request.setAttribute("attribute_name","value");
request.getAttribute("attribute_name"); // return the Object you have to
cast it
```

# Java & JEE Training

## Session tracking with servlets

MindsMapped Consulting

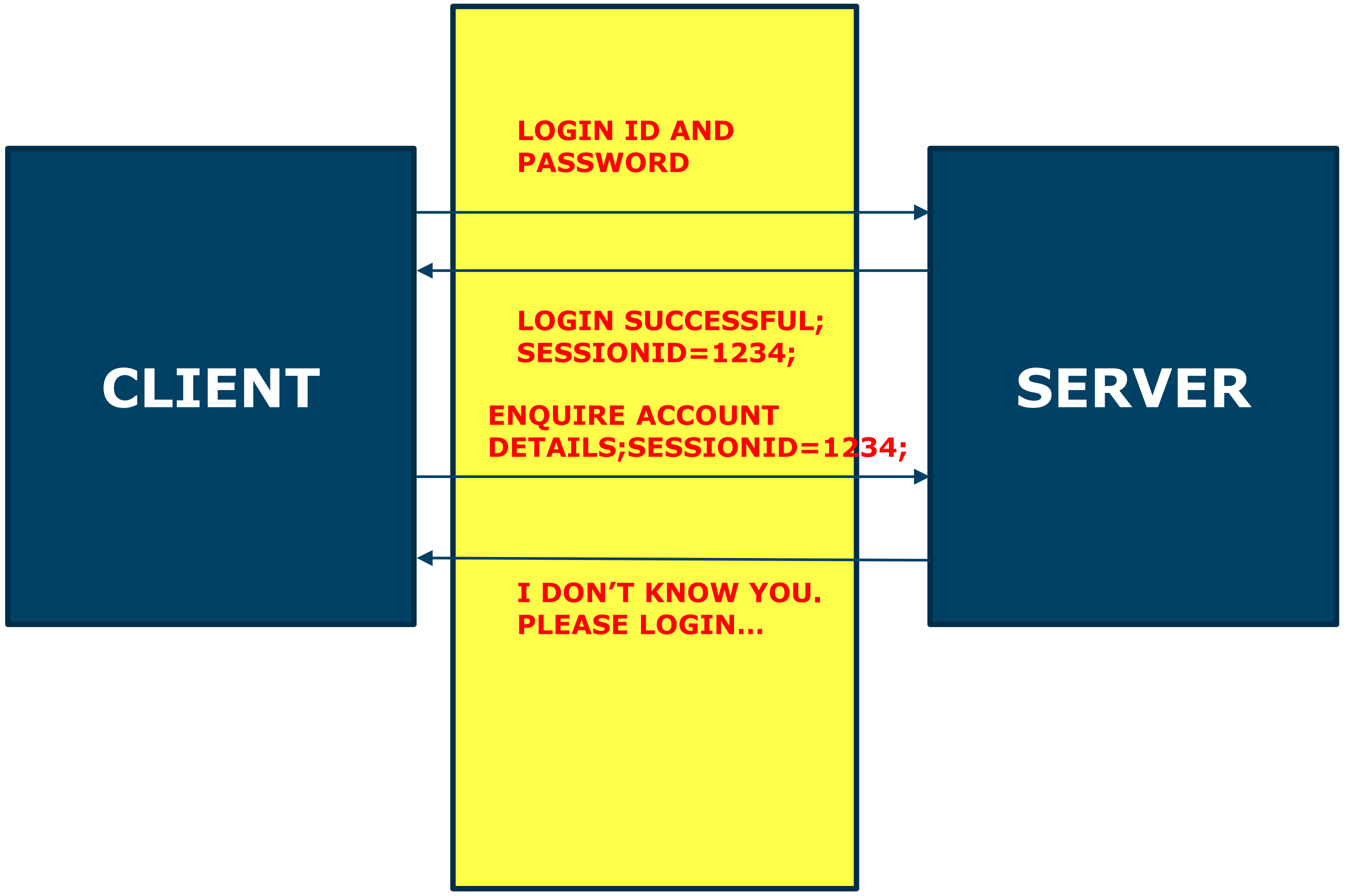
## HTTP IS STATELESS



**SESSION**
**- COOKIES…**

**HOW DO I MAINTAIN SESSIONS?**

## HTTP IS A STATELESS PROTOCOL


LOGIN ID AND PASSWORD
LOGIN SUCCESSFUL; SESSIONID=1234;
ENQUIRE ACCOUNT DETAILS;SESSIONID=1234;
I DON'T KNOW YOU. PLEASE LOGIN…
CLIENT
SERVER

SESSION
- COOKIES

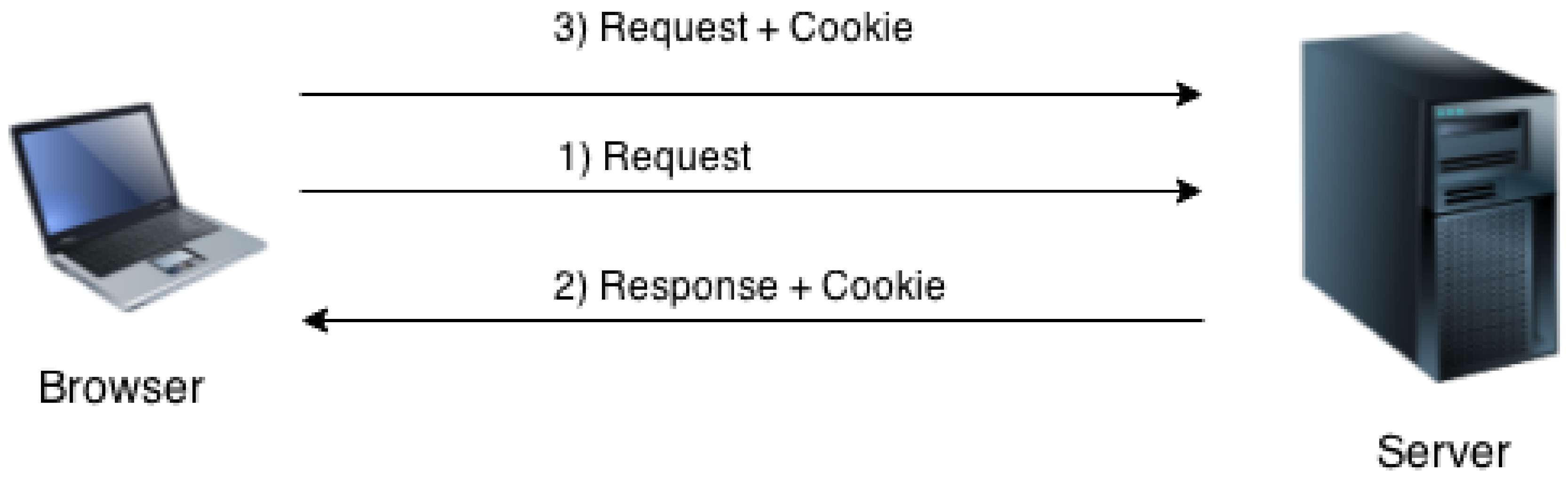
HOW DO I MAINTAIN SESSIONS?

## Why track sessions?

- HTTP is stateless protocol
- So we need to maintain state across multiple requests of a session using session tracking techniques.

- Session tracking techniques:
  - Cookies
  - Hidden Form Field
  - URL Rewriting
  - HttpSession

## How cookies work?



## Using cookies

- **javax.servlet.http.Cookie** class

- Creating a cookie:

```
Cookie ck=new Cookie("user",“pawan");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

- Deleting a cookie

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

- Get all cookies from request:

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing cookies info
}
```

## Example

- Demo of using cookies

## Hidden form fields…

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

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

- This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

- Advantage of Hidden Form Field
  - It will always work whether cookie is disabled or not.
- Disadvantage of Hidden Form Field:
  - It is maintained at server side.
  - Extra form submission is required on each pages.
  - Only textual information can be used.

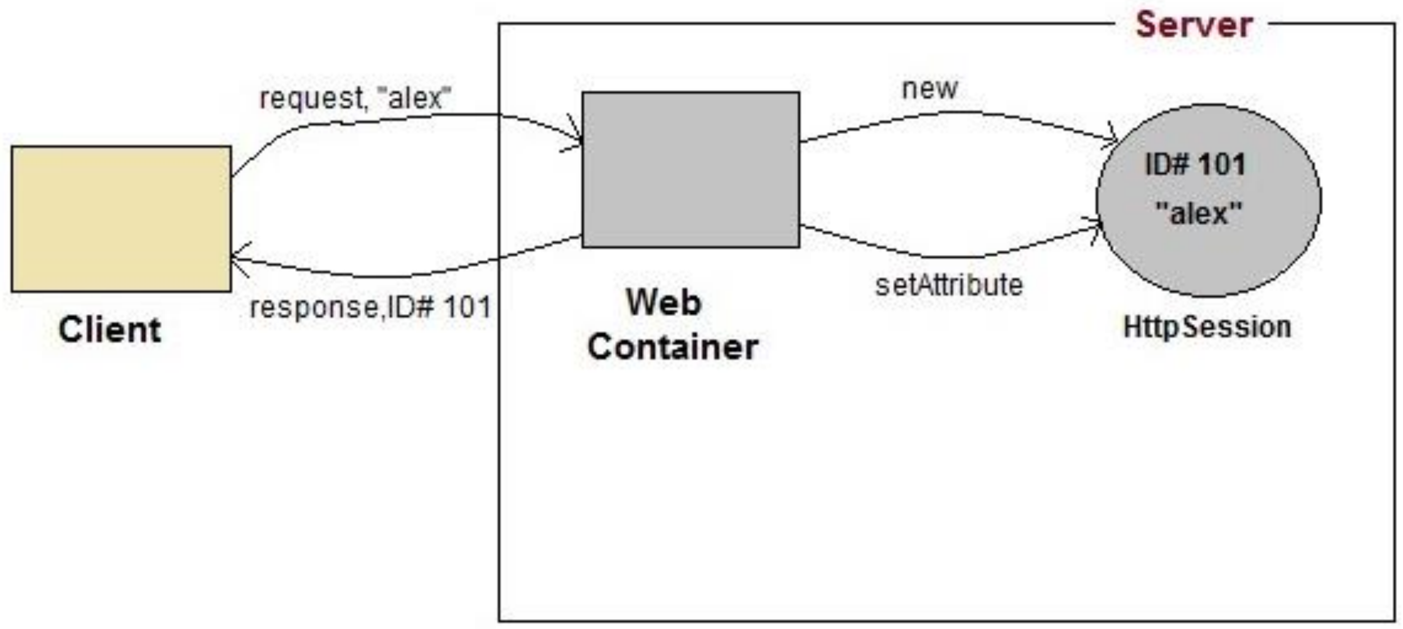
## URL Rewriting

- 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.
- For example, with http://mycompany.com/file.htm;sessionid=12345, the session identifier is attached as sessionid=12345 which can be accessed at the web server to identify the client.
- URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies but here drawback is that you would have generate every URL dynamically to assign a session ID though page is simple static HTML page.

- Advantage of URL Rewriting
  - It will always work whether cookie is disabled or not (browser independent).
  - Extra form submission is not required on each pages.
- Disadvantage of URL Rewriting
  - It will work only with links.
  - It can send Only textual information.

## Using HttpSession

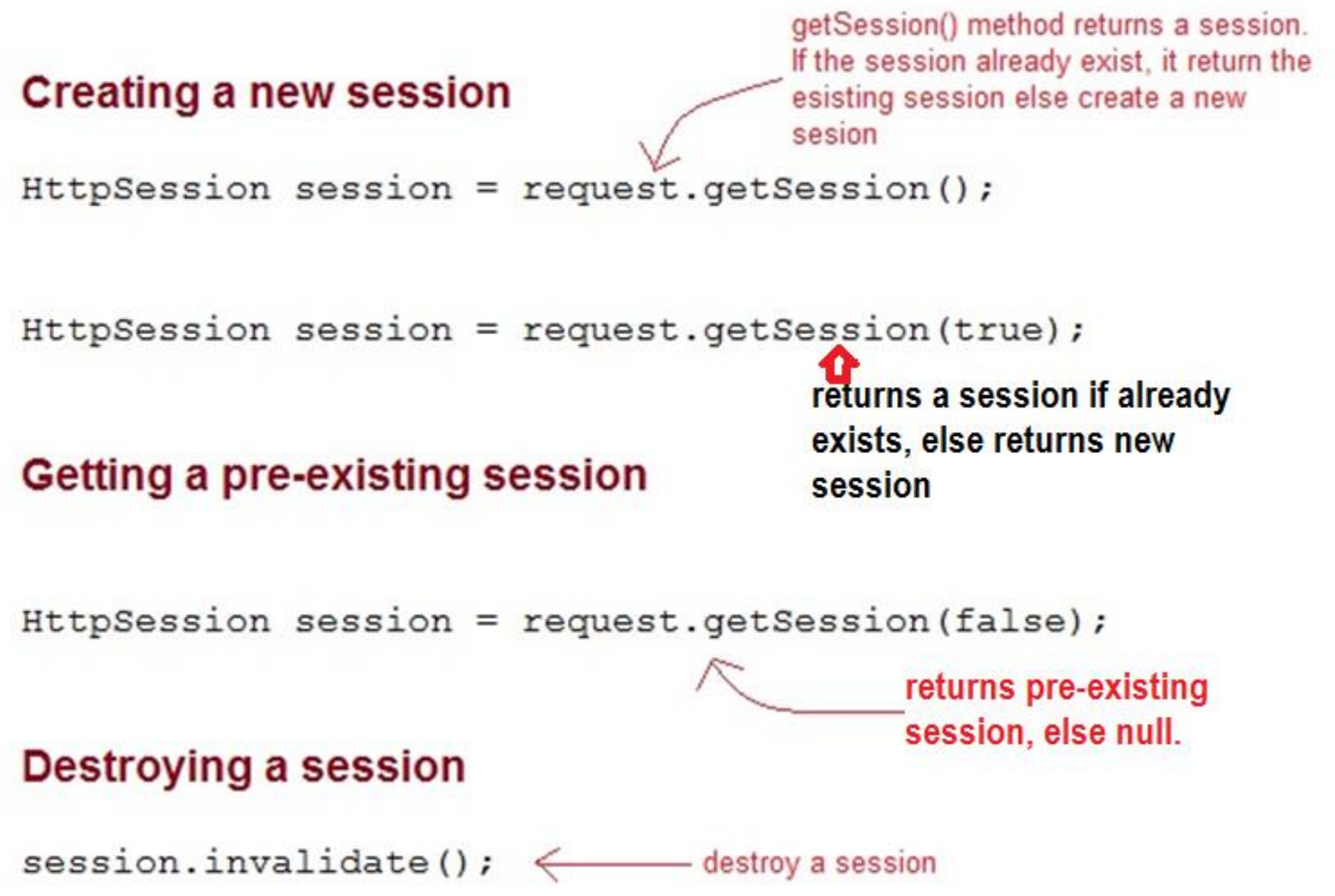
- HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from HttpSession object. Any servlet can have access to HttpSession object throughout the getSession() method of the HttpServletRequest object.



## Important HttpSession Methods

| Methods | Description |
|---|---|
| long getCreationTime() | returns the time when the session was created, measured in milliseconds since midnight January 1, 1970 GMT. |
| 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 |
| int getMaxInactiveInterval() | returns the maximum time interval, in seconds. |
| void invalidate() | destroy the session |
| boolean isNew() | returns true if the session is new else false |
| void setMaxInactiveInterval(int interval) | Specifies the time, in seconds,after servlet container will invalidate the session. |

## HttpSession usage



Creating a new session

getSession() method returns a session. If the session already exist, it return the esisting session else create a new sesion

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

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

returns a session if already exists, else returns new session

Getting a pre-existing session

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

returns pre-existing session, else null.

Destroying a session

```
session.invalidate();
```

destroy a session

## HttpSession Demo

- Demo
- Write a login / logout application using HttpSession