Java^{TM} Education & Technology Services

Building Dynamic Web Applications Using Servlets and JSP

Table of Contents

Chapter 1

Introduction To Servlet Technology

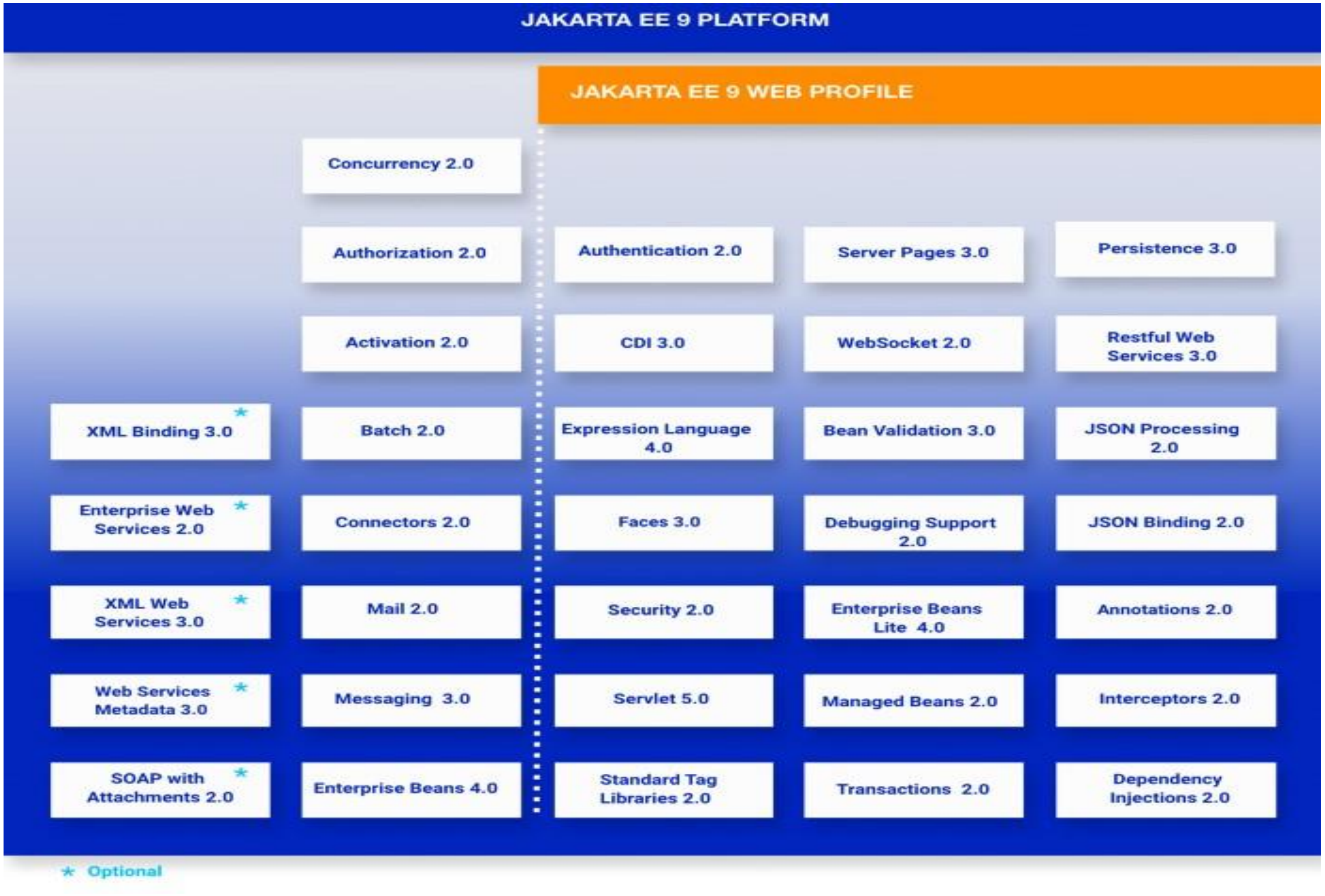
What is Servlet?

- Servlet API provides a way for developers to develop dynamic web Applications.
- Servlets Ver. 1.0 was released by Sun Microsystems in 1997.
- Servlet API was part of Java EE until version 8.
- And since then the API is moved to Jakarta EE

What is Servlet?

- Servlet 5.0 API is part of Jakarta EE 9 APIs released in 2020.
- API moved from javax.servlet to jakarta.servlet

What is Servlet?



What is Servlet?

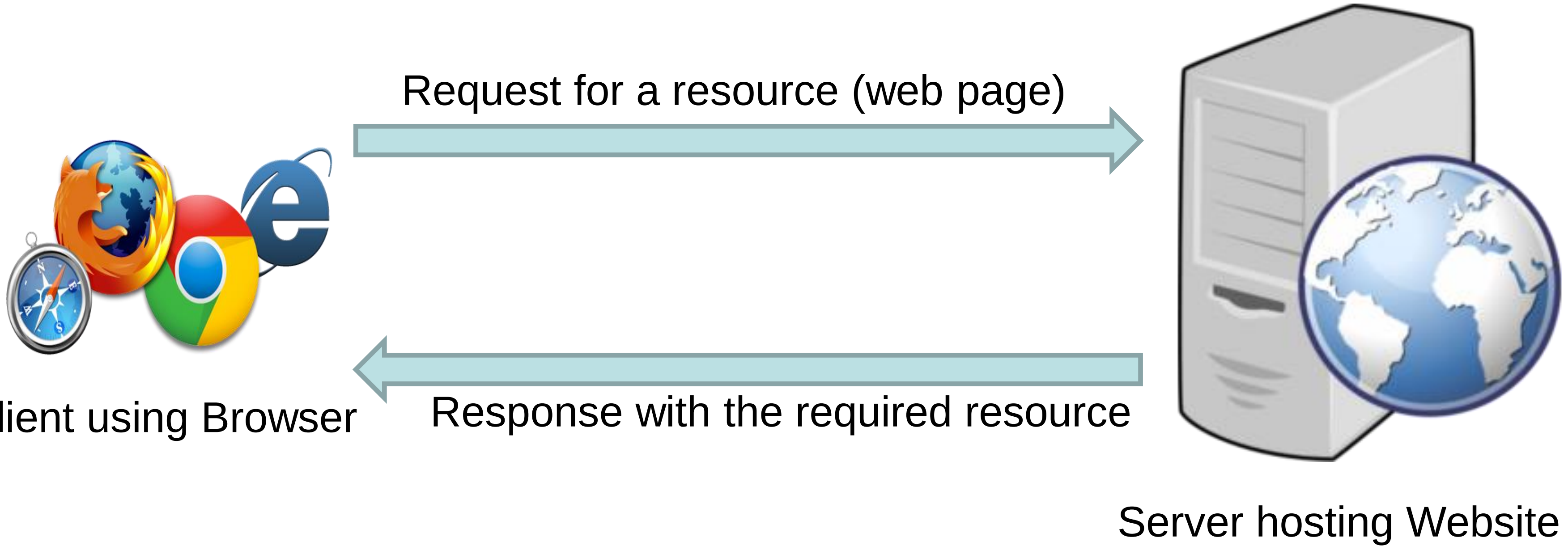
- Since **HTTP** is the foundation of data communication for the World Wide Web; so Servlets must know how to deal with it.

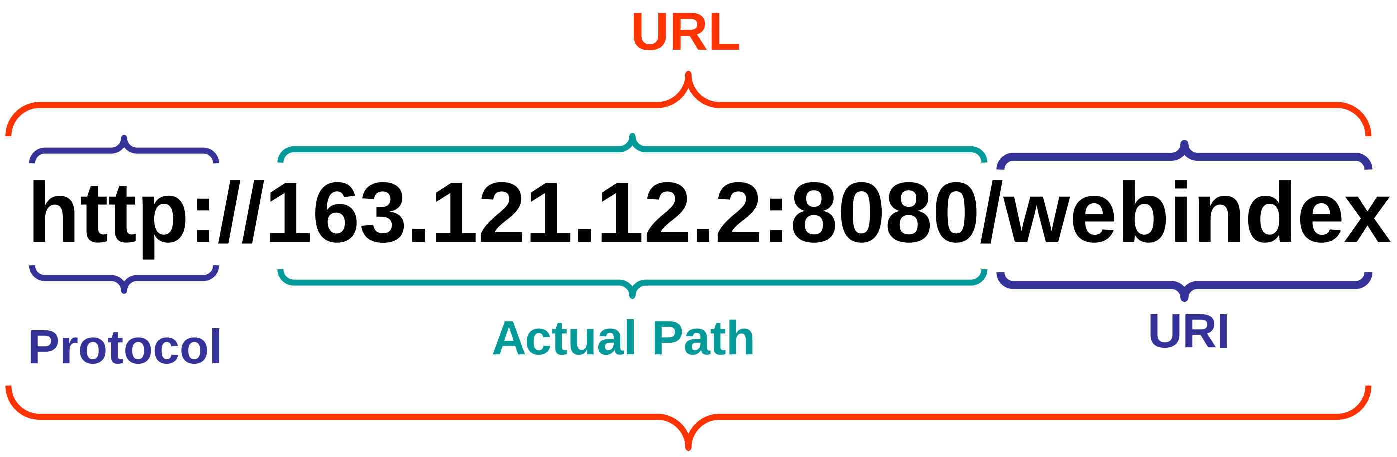
- Lets take a look at the HTTP first .

HTTP

- What is HTTP ?
  - Hyper Text Transfer Protocol.
  - Stateless Protocol.
  - It is an application layer protocol.
  - Working as request-response protocol

HTTP


Request for a resource (web page)
Client using Browser
Response with the required resource
Server hosting Website


URL
http://163.121.12.2:8080/webindex
Protocol
Actual Path
URI

HTTP Request

- HTTP Request consists of 3 components:

  1.Request-Line

  Method / Request-URI / Protocol-Version

  – Methods could be : Get ,Post, Head, Put, Trace, Options, Delete

- Example:
  – Get /servlet /default.jsp HTTP/1.1
  – Post / servlet /index.jsp   HTTP/1.1

HTTP Request (cont'd)

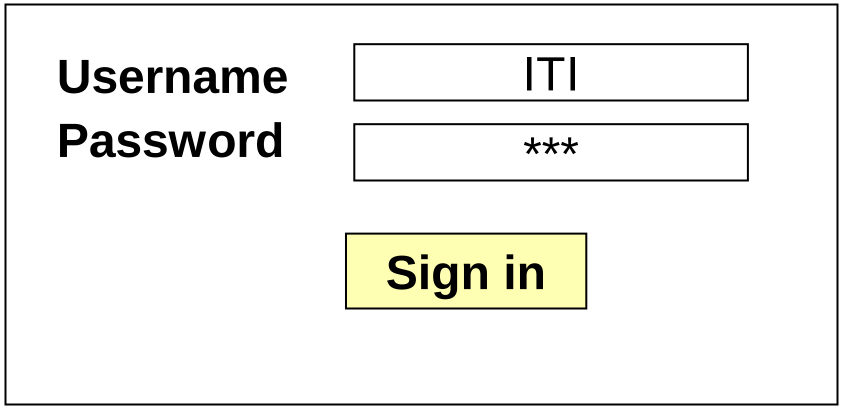
2.Request Headers

- Indirectly set by the browser and sent immediately after the request line. The following are examples of request headers:
  - **Accept**: text/plain ;text/html (Note : type/* , */*)
  - **Accept-language**: en-ar
  - **Accept-Encoding** : gzip
  - **Connection** : keep- Alive
  - **Host** : www.iti.gov.eg or localhost:8080
  - **Referer** : http://www.mcit.gov.eg/main.htm
  - **User-agent** : Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
  - **Content-length** : 33 *(in case of POST)*
  - **If-Modified-Since**: specified date
  - **If-Unmodified-Since** : Sat, 29 Oct 1994 19:43:31 GMT
  - **Cookie**: userID=id456578

HTTP Request (cont'd)

3. Entity Body (in case of POST method):

UserName=ITI & Password =iti



- Note: If the GET method was used:
  - There will be no entity body.
  - URL in the address bar of the browser will look something like:

www.sitename.com/MyServlet? userName=ITI&password=iti

HTTP Response

- HTTP Response consists of 3 components:

  1. Status-Line

  Protocol-Version / Status Code / Description

  - Status Code Ranges:
    - 100-199 informational, the client must respond with a action
    - 200-299 : request is successful
    - 300-399 : for moving files , it includes a location header indicating the new address
    - 400-499 :indicates an error by the client
    - 500-599 : indicates an error by the server

  – **Example**

  - HTTP/1.1 200 ok
  - HTTP/1.1 404 error

HTTP Response (cont'd)

2.Response Headers

Server: Tomcat/5.5

Date: Mon, 3 Jan 2006 13:13:33 GMT

Content-Type: text/html

Last-Modified: Mon, 11 Jan 2005 13:23:42 GMT

Content-Length: 112

Content-Encoding = gzip

HTTP Response (cont’d)

3.Entity Body

```
<HTML>
  <HEAD>
        <TITLE>HTTP Response Example
        </TITLE>
  </HEAD>
  <BODY>
        Welcome to servlets and jsp course ☺
  </BODY>
</HTML>
```

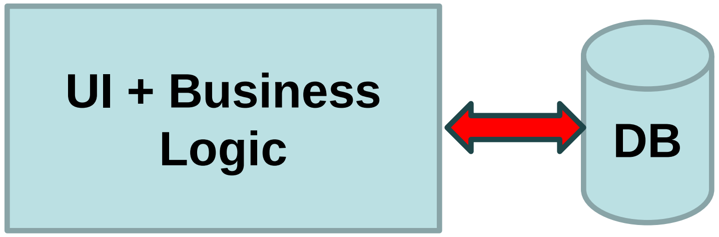
What is Servlet?

Now Back To Servlet again:

we can say that **A Servlet** is a Java program that runs on a **server**, that produces dynamic pages typically in **response** to client **requests**.

Application Architecture

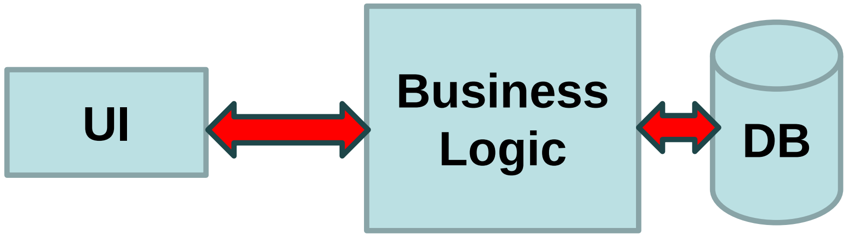
In desktop applications



Application on all users machines

What is so called Two Tier Architecture

In Web applications



User's web Browser

Server Hosting Web Application(Servlets)

What is so called Three Tier Architecture

Types of Servers

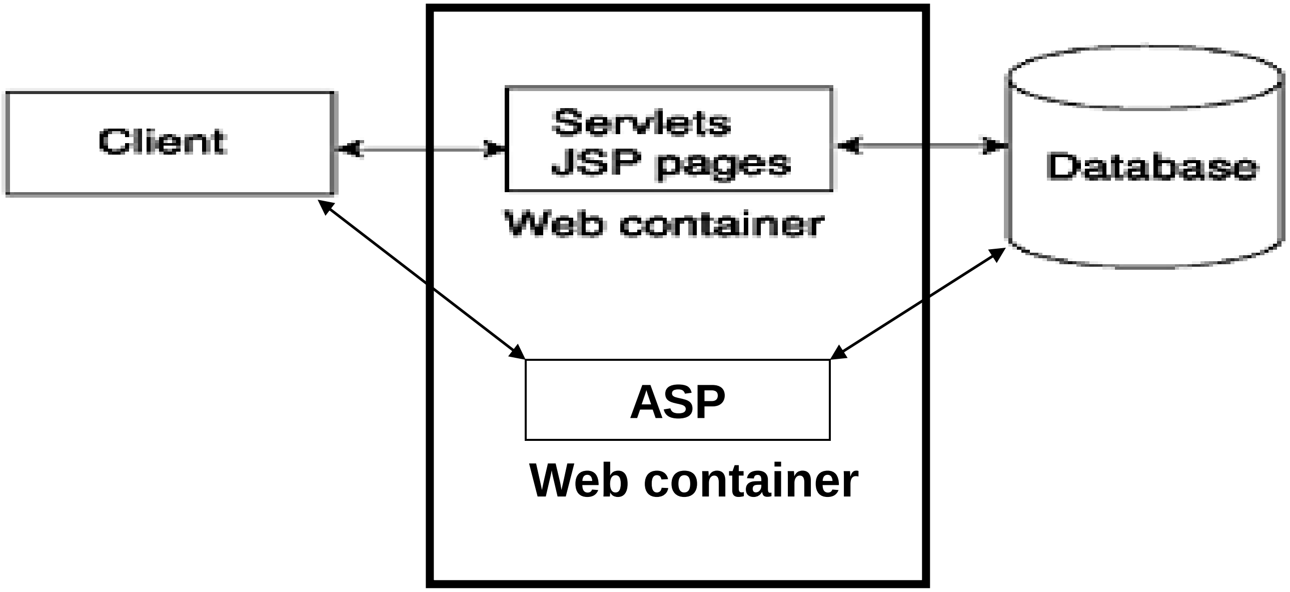
There are two types of server that can host Web application written in Servlet

1. Web Servers

2. Application Servers

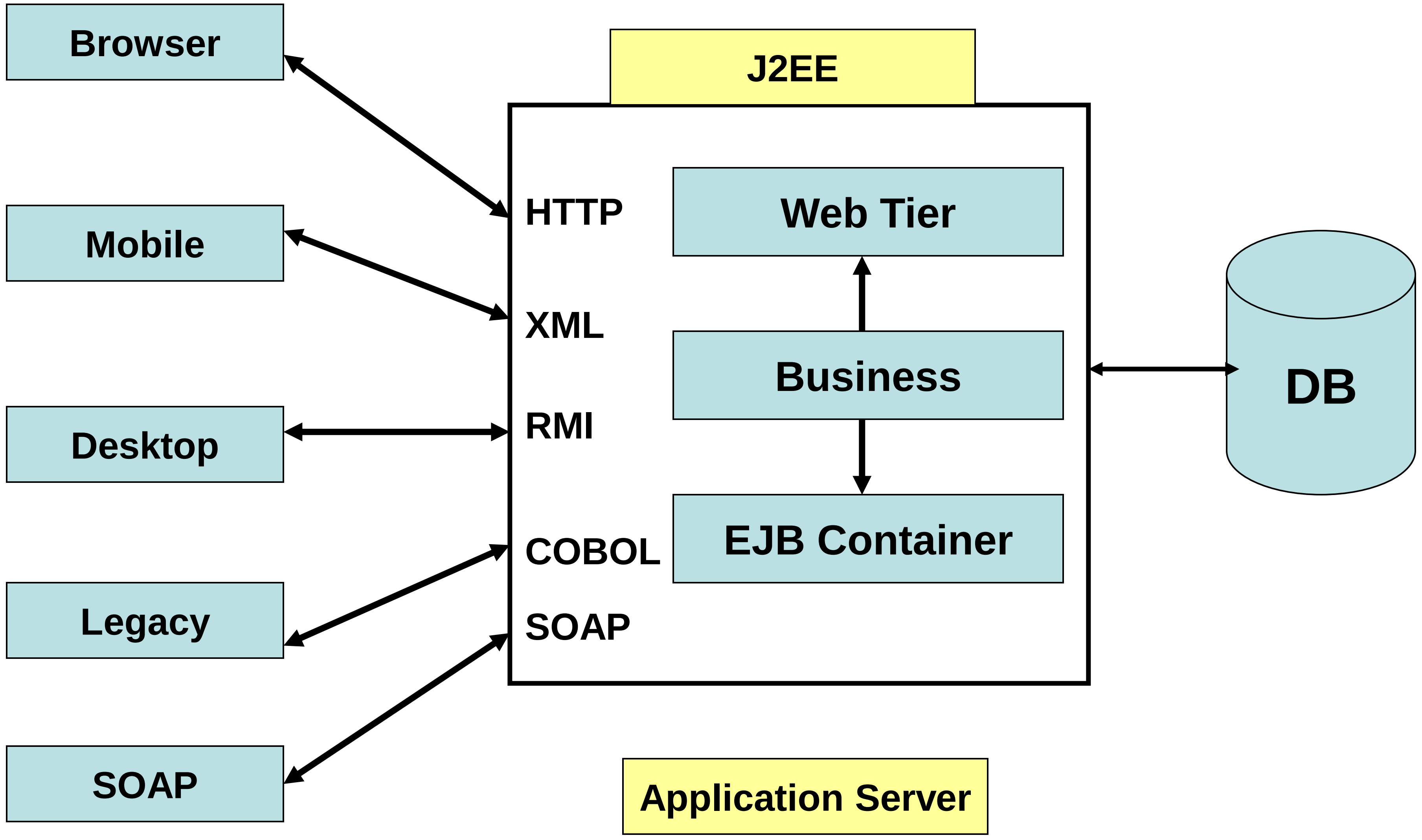
Types of Servers (cont'd)

- Web Server :



Web Server

Types of Servers (cont'd)



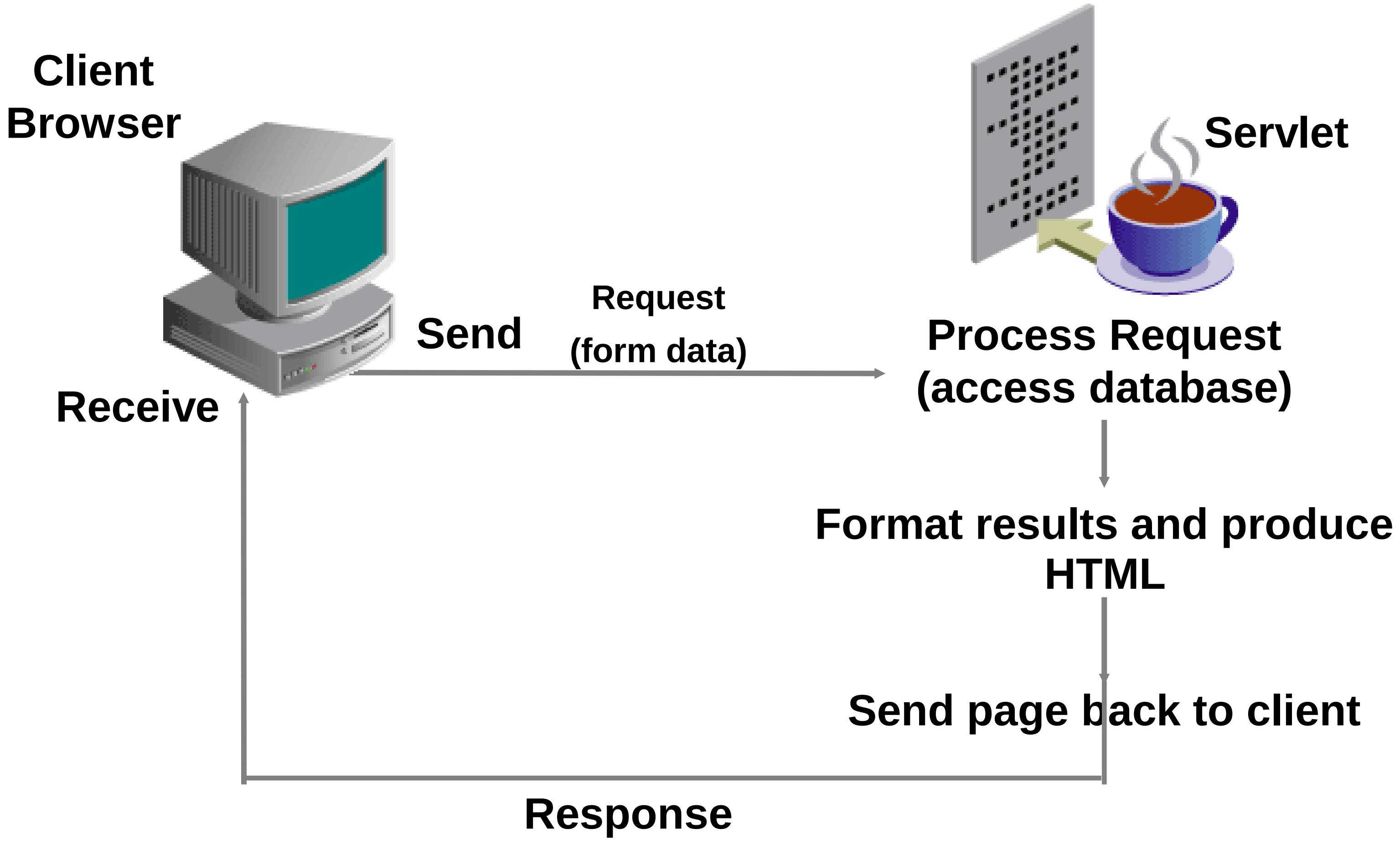
Web Technologies

- CGI (Common Gateway Interface)
- Cold Fusion
- Server Side Java Script (SSJS)
- PHP
- Servlet
- JSP
- ASP
- ASP.NET

Chapter 2

How Servlets works?

How Servlets works?



How Servlets works?(cont’d)

- Servlet main jobs:
  - It reads and process data sent by the client.
  - Send the data   to the client with a proper format.

- It’s not restricted to HTTP requests

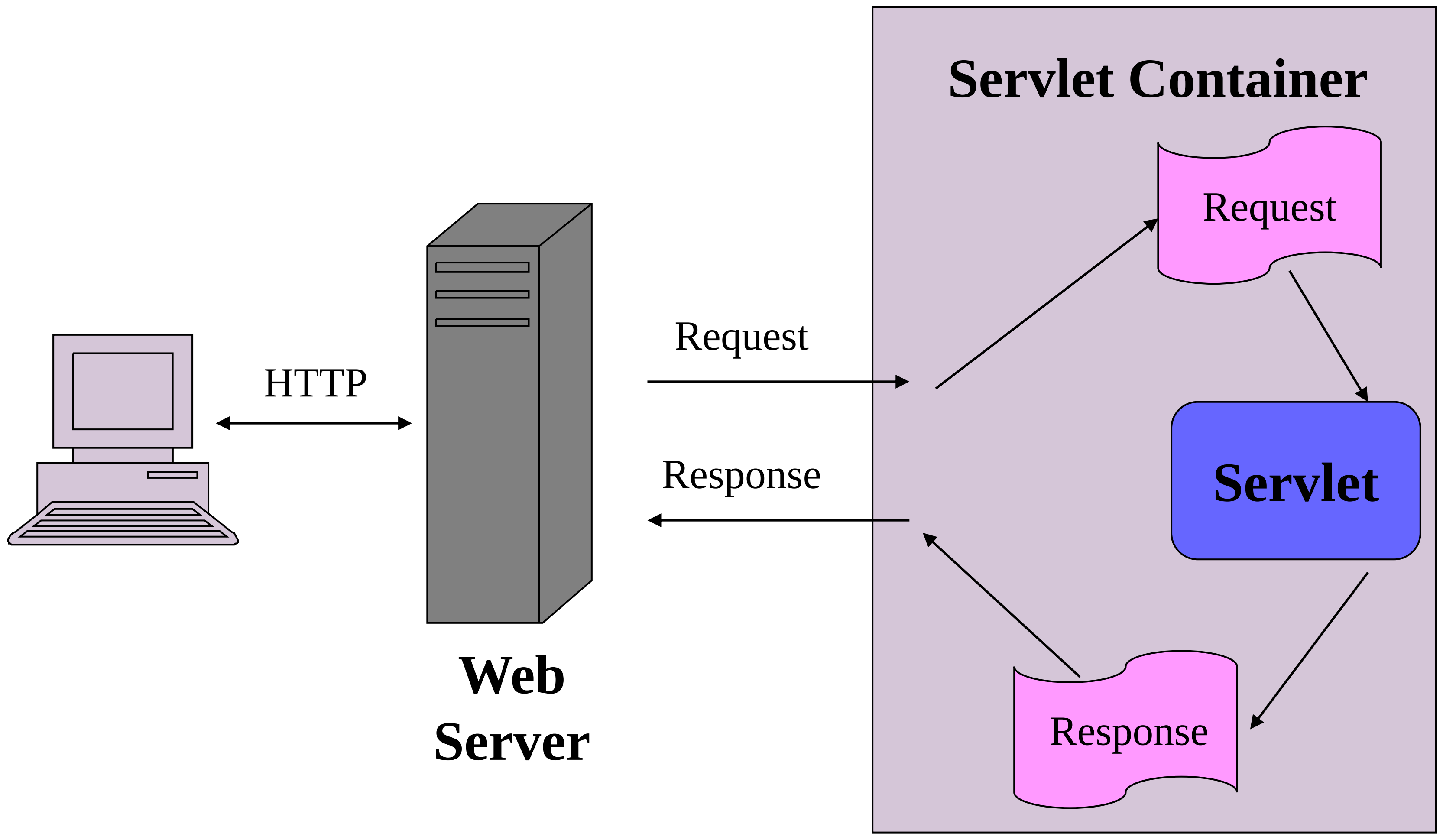
How Servlets works?(cont’d)

Since Servlets is a Java Class it needs something to be responsible of the following:

- loading
- Instantiating
- unloading.
- Managing servlet’s life cycle.
- creates and manages request and response objects

And this is the job of the **Servlet Container**

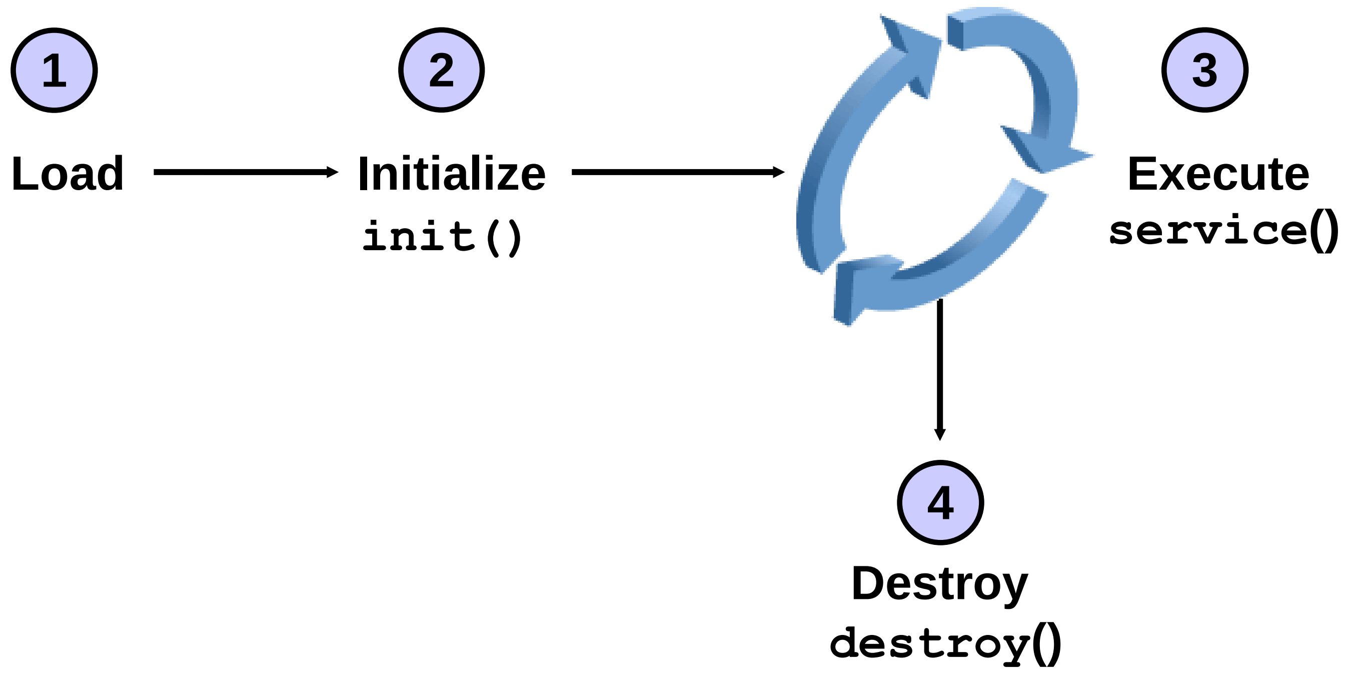
Servlet Container



Servlet Container (cont’d)

- Servlet containers:
  - Apache Tomcat
  - GlassFish
  - WebSphere
  - JBoss
  - WebLogic

Servlets Life Cycle



Benefits of Servlets

- Performance
- Portability - Widespread acceptance
- Rapid development cycle
- Robustness
- Secure
- Inexpensive

Web application structure

- Any servlet/jsp application must contains the folder named WEB-INF which may contains:
  - **classes** folder: representing the servlets after compilation (only if the application contains servlets)
  - Deployment descriptor (**web.xml**)
  - **lib** folder
- Also the web application may optionally contains:
  - **images, html , tld and jsp** pages

What is the deployment descriptor ?

- Contains meta-data for a Web Application and that is used by the container when loading a Web Application.
  - It identifies all the elements contained in the web application for the servlet container to be able to know them
  - It maps them into URL patterns to be able to call them as clients
  - It contains configuration settings specific to that application.

Example

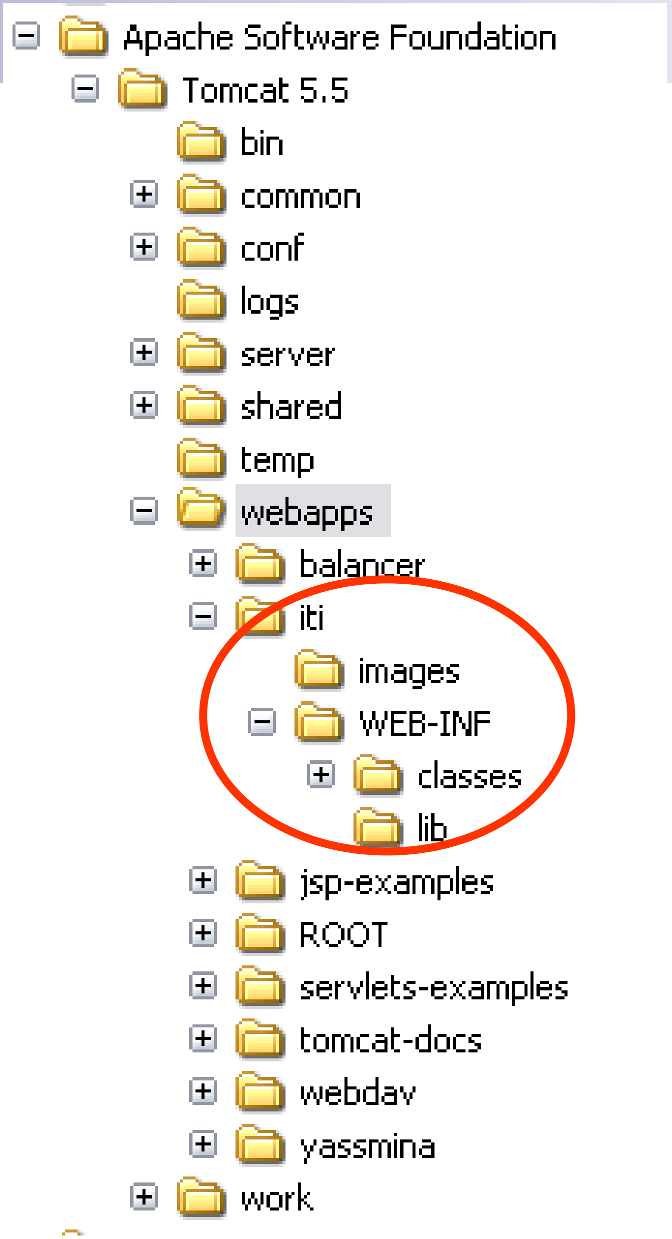
```
<web-app>

    <servlet>
        <servlet-name>Testing</servlet-name>
        <servlet-class>TestingServlet</servlet-class>
    </servlet>

    <servlet-mapping>
            <servlet-name>Testing</servlet-name>
            <url-pattern>/hi</url-pattern>
        </servlet-mapping>

</web-app>
```

Tomcat setup and structure

Making Your First Servlet

There are six steps to do so :

1. Create a directory structure under tomcat for your application



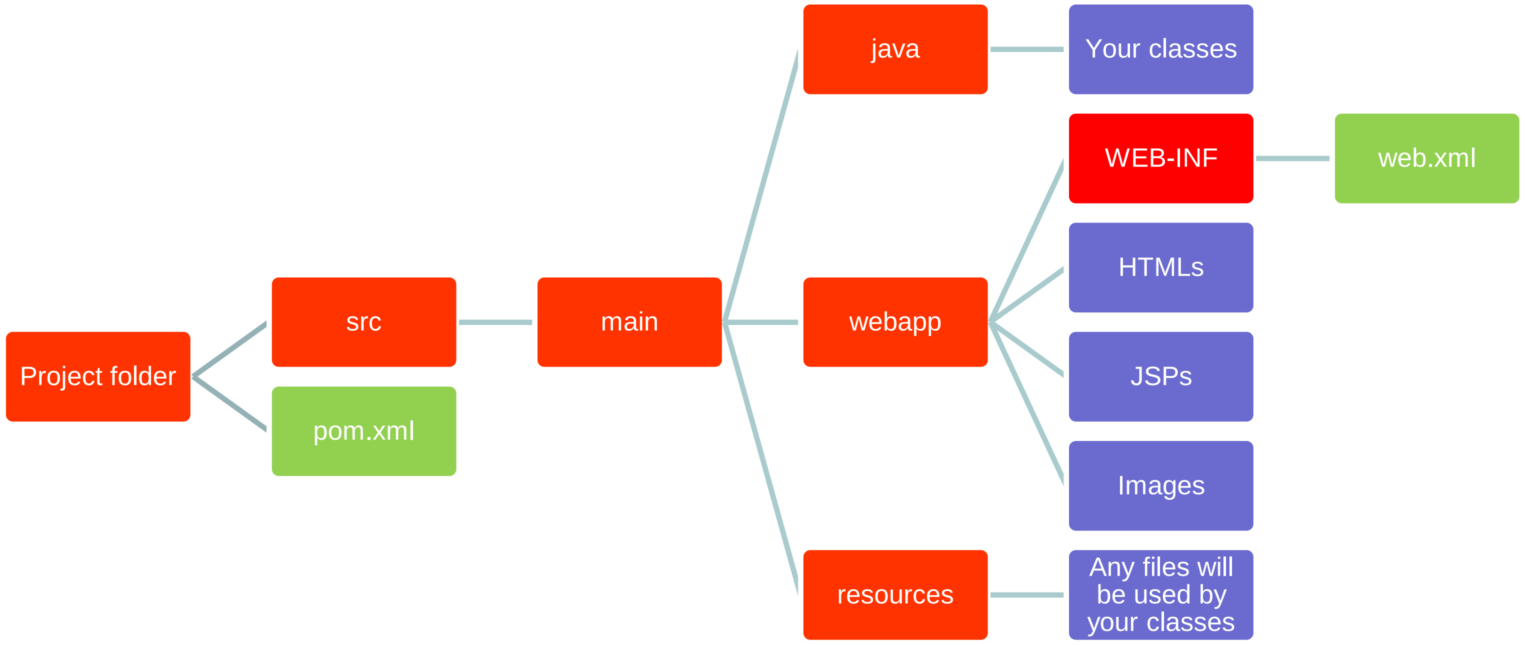
Making Your First Servlet (cont’d)

2. Write your servlet Code
3. Compile your servlet
   - Remember that we have to include the servlet and jsp libraries to the class path
   - javac -classpath C:…\tomcat\common\lib\servlet.jar TestingServlet.java
   - Or may it permanently in the classpath environment variable
   - Or put it in jdk/jre/lib/ext

Making Your First Servlet (cont'd)

or you can simply use Maven

Now the folder structure of the project will look like this:



Making Your First Servlet (cont'd)

Now in pom.xml use war for packaging

```
<packaging>war</packaging>
```

Use the following dependency for Jakarta EE 9

```
<dependency>
     <groupId>jakarta.platform</groupId>
    <artifactId>jakarta.jakartaee-api</artifactId>
    <version>9.0.0</version>
    <scope>provided</scope>
</dependency>
```

Making Your First Servlet (cont’d)

Then add the following plugins

This one for compiling

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <source>11</source>
          <target>11</target>
        </configuration>
</plugin>
```

Making Your First Servlet (cont'd)

This one for generating war file

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.0</version>
        <configuration>
            <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
 </plugin>
```

Making Your First Servlet (cont'd)

And you can use this plugin for deploying:

```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
                <username>admin</username>
                <password>admin</password>
                <url>http://localhost:9191/manager/text</url>
                <path>/test</path>
        </configuration>
</plugin>
```

The credentials of the tomcat user with "manger-script" and "manger-gui" roles

The URL of you tomcat server, so the plugin can deploy the application

The context name that will be used to access the application after deployement

Making Your First Servlet (cont'd)

```
<web-app xmlns= "https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation= "https://jakarta.ee/xml/ns/jakartaee
                    https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
  version= "5.0"
  metadata-complete= “false">

        <servlet>
            <servlet-name>Testing</servlet-name>
            <servlet-class>MyFirstServlet</servlet-class>
        </servlet>

        <servlet-mapping>
              <servlet-name>Testing</servlet-name>
              <url-pattern>/MyTest</url-pattern>
          </servlet-mapping>


</web-app>
```

Making Your First Servlet (cont'd)

5. Run the Tomcat
6. If you are using maven run the command

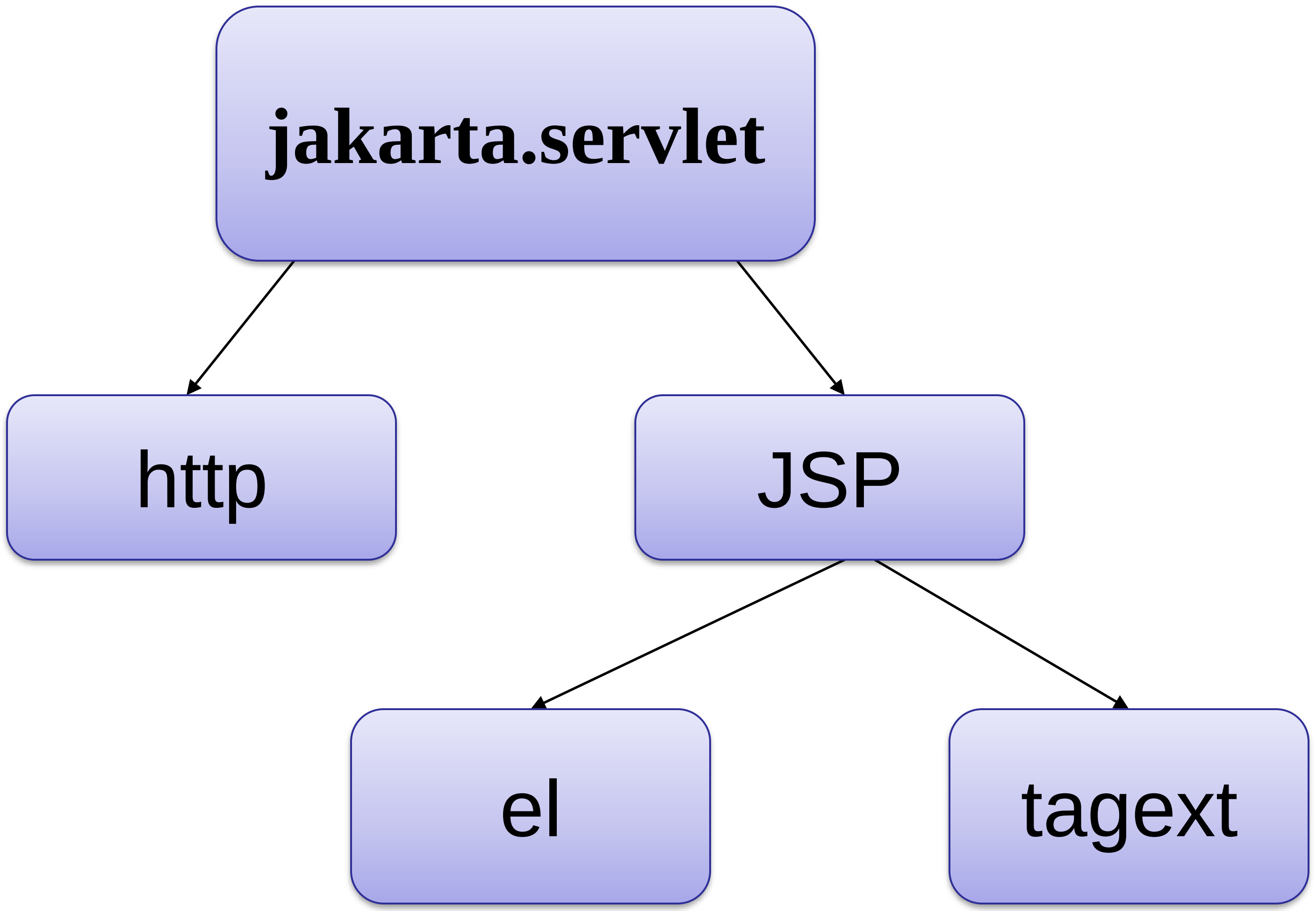
   mvn install tomcat7:deploy

5. Call your Servlet from the Web Browser
   - http://domain-name/context name/servlet-name
   - Example:
     - http://localhost:9191/test/MyTest

Chapter 3

How to write servlets code?

Servlet Package Structure



jakarta.servlet.*

- Some basic interfaces :
  - Servlet
  - ServletConfig
  - ServletContext
  - ServletRequest
  - ServletResponse
  - SingleThreadModel
  - Request Dispatcher

jakarta.servlet.*

- Some basic interfaces :
  - Servlet
  - ServletConfig
  - ServletContext
  - ServletRequest
  - ServletResponse
  - SingleThreadModel
  - Request Dispatcher

jakarta.servlet.* (cont’d)

- Some basic classes :
  - GenericServlet
  - ServletInputStream
  - ServletOutputStream
- The Exception Classes are:
  - ServletException
  - UnavailableException

Servlet Interface

- Servlet : it’s the basic interface , any servlet must implements it directly or indirectly
- It consists of 5 main methods :
  - init (ServletConfig config)
  - service(ServletRequest , ServletResponse)
  - destroy()
  - getServletConfig()
  - getServletInfo()

Servlet Example

- The following code is a typical implementation of the interface:

```
import jakarta.servlet.*;
import java.io.*;
public class MyServlet implements Servlet
{
   public void init(ServletConfig config) throws ServletException
   {
       System.out.println("I am inside the init method");
   }
   public void service(ServletRequest request,
                        ServletResponse response)
                          throws ServletException, IOException
    {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
      out.println(“<br>Welcome to Servlets and JSP Course");
      System.out.println("I am inside the service method");
   }
```

Write to the response

Servlet Example cont’d

```
   public void destroy()
   {
        System.out.println(“I am inside the destroy method");
   }

   public String getServletInfo()
   {
        return null;
   }

   public ServletConfig getServletConfig()
   {
        return null;
   }
}
```

SingleThreadModel Interface

- The `SingleThreadModel` interface is used to prevent simultaneous access to instance variables by multiple threads.
- Only one request thread accesses a single instance of the servlet at a time.
- The server queues the requests and passes them one at a time to a single servlet instance.
- Or the server creates a pool of multiple instances, each handles one request at a time.(however, unsafe).

SingleThreadModel Interface cont'd

- It became deprecated starting servlet 2.4
- It caused latency and bad performance in case of high traffic servlets.
- Instead, the better approach is to synchronize the part of the code that manipulates the shared data:

```
synchronized(this)
{
  String id = "User-ID-" + nextID;
  out.println(id);
  nextID = nextID + 1;
}
```

ServletConfig Interface

- The **`ServletConfig`** interface is used by a servlet to obtain some of the configuration written specifically for it in the deployment descriptor file (web.xml).
- It is passed by the servlet container to the **`init(..)`** method as parameter.
- Such configuration information is better written in the xml file instead of hard coding it inside the servlet's code.
- Such approach makes it easy to alter such information after the servlet's source code has been compiled.
- Each servlet is passed its own set of configuration parameters.

ServletConfig Interface cont’d

- Configuration parameters (init params) can be written inside the web.xml file as follows:

```
<servlet>
      <servlet-name>MyServlet</servlet-name>
      <servlet-class>ServletConfigTest</servlet-class>
      <init-param>
            <param-name>AdminEmail</param-name>
            <param-value>admin@iti.gov</param-value>
      </init-param>
</servlet>
```

- The whole <init-param> tag is repeated for each parameter.

ServletConfig Interface cont’d

- The **`ServletConfig`** interface has 4 methods:
  - **`Enumeration getInitParameterNames()`**
  - **`String getInitParameter(String paramName)`**
  - **`String getServletName()`**
  - **`ServletContext getServletContext()`**

ServletConfig Example

- The following code demonstrates the use of the config:

```
public class ServletConfigTest implements Servlet
{
   ServletConfig myConfig;

   public void init(ServletConfig config) throws ServletException
   {
       myConfig = config;
   }

    public void service(ServletRequest request,
                              ServletResponse response)
                                throws ServletException, IOException
    {
       String str = myConfig.getInitParameter(“AdminEmail”);
       PrintWriter out = response.getWriter();
       out.println (“The administrator email is: ” + str);
   }
}
```

Lab Exercise

Assignment

- Use servletConfig to pass variables from web.xml to the servlet and display it in the response.
  - Like DB connection Statement.
  - DB user name and password.

```
<init-param>
                    <param-name> DatabaseAddress </param-name>
                    <param-value> 163.121.12.25 </param-value>
</init-param>
<init-param>
                    <param-name> userName </param-name>
                    <param-value> iTi </param-value>
</init-param>
<init-param>
                    <param-name> password </param-name>
                    <param-value> iTi </param-value>
</init-param>
```

The ServletContext Interface

- provides information to servlets about the environment in which they are running.
- It represents a context within which a Web application executes.
- Each web application has only one ServletContext object.
- Defines a set of methods that a servlet uses to communicate with its servlet container.

The ServletContext Interface (cont'd)

- How to get Servlet Context of the servlet?
  - Through the ServletConfig , using the method "getServletContext()".
  - The ServletContext object gives you the ability to :
    - bind an object attribute into the context by name. which will be available to any other servlet that is part of the same web application.
    - Read context initialization parameter from the <context-param> tag written in the web.xml

ServletContext Interface (cont'd)

- Context parameters of a web application (context params) can be written inside the web.xml file as follows:

```
<web-app>
        <context-param>
                <param-name>DollarExchangeRate</param-name>
                <param-value>17.5</param-value>
        </context-param>
      .
      .
      .
      .
</web-app>
```

- The whole <context-param> tag is repeated for each parameter. It should be placed at the beginning of the file.

ServletContext Interface cont’d

- Commonly used methods:
  - `Enumeration getInitParameterNames()`
  - `String getInitParameter(String paramName)`
  - `Enumeration getAttributeNames()`
  - `Object getAttribute(String attName)`
  - `void setAttribute(String attName, Object value)`
  - `void removeAttribute(String attName)`
  - `void log(String msg)`

Example

```
public class MyservletContext implements Servlet  {
        ServletConfig sConf;

        public void init (ServletConfig config) throws ServletException {
            sConf = config;
        }

        public void service (ServletRequest request,ServletResponse
            response)throws ServletException, IOException
        {
                ServletContext servletContext = sConf.getServletContext();
                PrintWriter  out = response.getWriter();
                String str= servletContext. getInitParameter("DollarExchangeRate");
                out.println("DollarExchangeRate is: " + str);
        }

}
```

Example

Setter Servlet

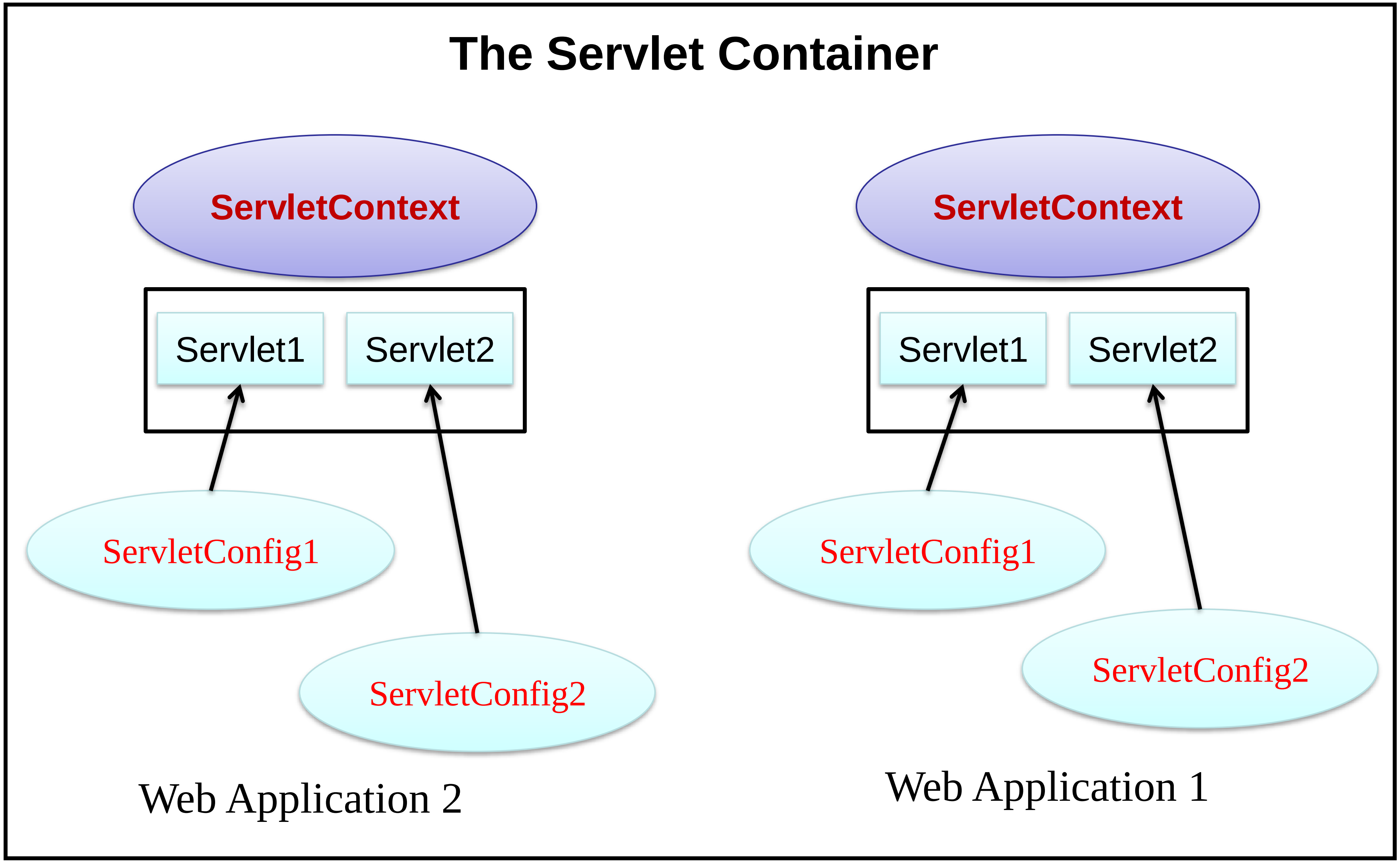
```
public class MyContextSetter implements Servlet  {
        ServletConfig sConf;

        public void init (ServletConfig config) throws ServletException {
            sConf = config;
        }

        public void service (ServletRequest request,ServletResponse
            response)throws ServletException, IOException
        {
                ServletContext servletContext = sConf.getServletContext();

                servletContext.setAttribute(“MyPassword”,”iti”);
        }

}
```

Example

Getter Servlet

```
public class MyContextGetter implements Servlet  {
        ServletConfig sConf;

        public void init (ServletConfig config) throws ServletException {
            sConf = config;
        }

        public void service (ServletRequest request,ServletResponse
            response)throws ServletException, IOException
        {
                ServletContext servletContext = sConf.getServletContext();

                PrintWriter  out = response.getWriter();

                String str=(String) servletContext.getAttribute(“MyPassword”);

                out.println(“MyPassword is: " + str);
        }

}
```

Config & Context



Lab Exercise

Assignment

- Sharing info between servlets: A servlet will assign a username and password attributes , and another servlet will display it
- Your servlets should read the initialized parameters given to them by <context-param> tag in web.xml

```
<context-param>
		<param-name>Country</param-name>
		<param-value>Egypt</param-value>
</context-param>
```

ServletRequest Interface

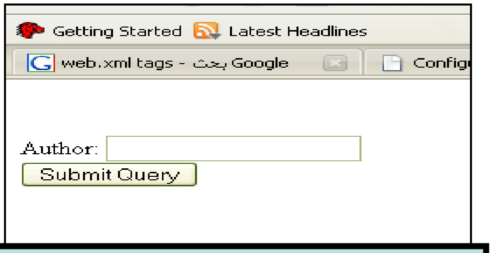
- The **`ServletRequest`** defines an object that is used to hold information coming from the user's request, including parameter name-value pairs, attributes, and an input stream.

- Commonly used methods:
  - **`Enumeration getParameterNames()`**
  - **`String getParameter(String paramName)`**
  - **`String[] getParameterValues(String paramName)`**
  - **`String getRemoteHost()`**

Sending Request Parameters to a Servlet

- The <form> tag in HTML is used as follows:

```
<html>
  <head>
    <title> Sending parameters in the request </title>
  </head>

  <body>
    <form ACTION=MyFirstServletURL METHOD="Get">
        Author : <Input TYPE="text" NAME="MyName">
        < Input TYPE="SUBMIT" VALUE="Submit">
        < Input TYPE="RESET" VALUE="Reset">
    </form >
  </body >
</html >
```



GET /MyFirstServlet HTTP/1.1 ? MyName="ITIStudent"

Sending Request Parameters to a Servlet cont’d

- In order for the servlet to get the parameters from the request, the following code is written in the **`service(..)`** method:

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

   PrintWriter out = response.getWriter();
   String str = request.getParameter(“MyName");
   out.println(“The user input is: “ + str);
 }
```

ServletResponse Interface

- The ServletResponse interface defines an object that carries the server's response to the user (i.e. back to the client's browser).

- Commonly used methods:
  - **`PrintWriter getWriter()`**
  - **`setContentType(String mimeType)`**
  - **`setContentLength(int len)`**

ServletResponse Example

- Sending response in Excel Sheet Format:

```
public void service(ServletRequest request,
                          ServletResponse response)
                            throws ServletException, IOException
   {
       response.setContentType("application/vnd.ms-excel");
       PrintWriter out = response.getWriter();
       out.println("\t jan \t feb \t march \t total");
       out.println("salary \t100 \t200 \t300 \t=sum(B2:D2)");
   }
```

GenericServlet Wrapper Class

- The **`GenericServlet`** class is a convenient class that implements the **`Servlet`** interface.

- It overrides the required 5 methods, and hence makes your code simpler (i.e. you extend the class and only override the necessary methods).

Lab Exercise

Response in Different Formats

- Develop different servlets that send response in different mime types (e.g. excel sheet, word document, …etc).

- Hint: Search on the internet for names of famous mime types.

Chapter 4

Getting Familiar with HTTP Package

jakarta.servlet.http.*

- HTTPServlet ← GenericServlet
- HTTPServletRequest ← ServletRequest
- HTTPServletResponse ← ServletResponse

jakarta.servlet.http.* (cont'd)

- Some basic interfaces
  - HttpServletRequest
  - HttpServletResponse
  - HttpSession
- Classes
  - Cookie
  - HttpServlet

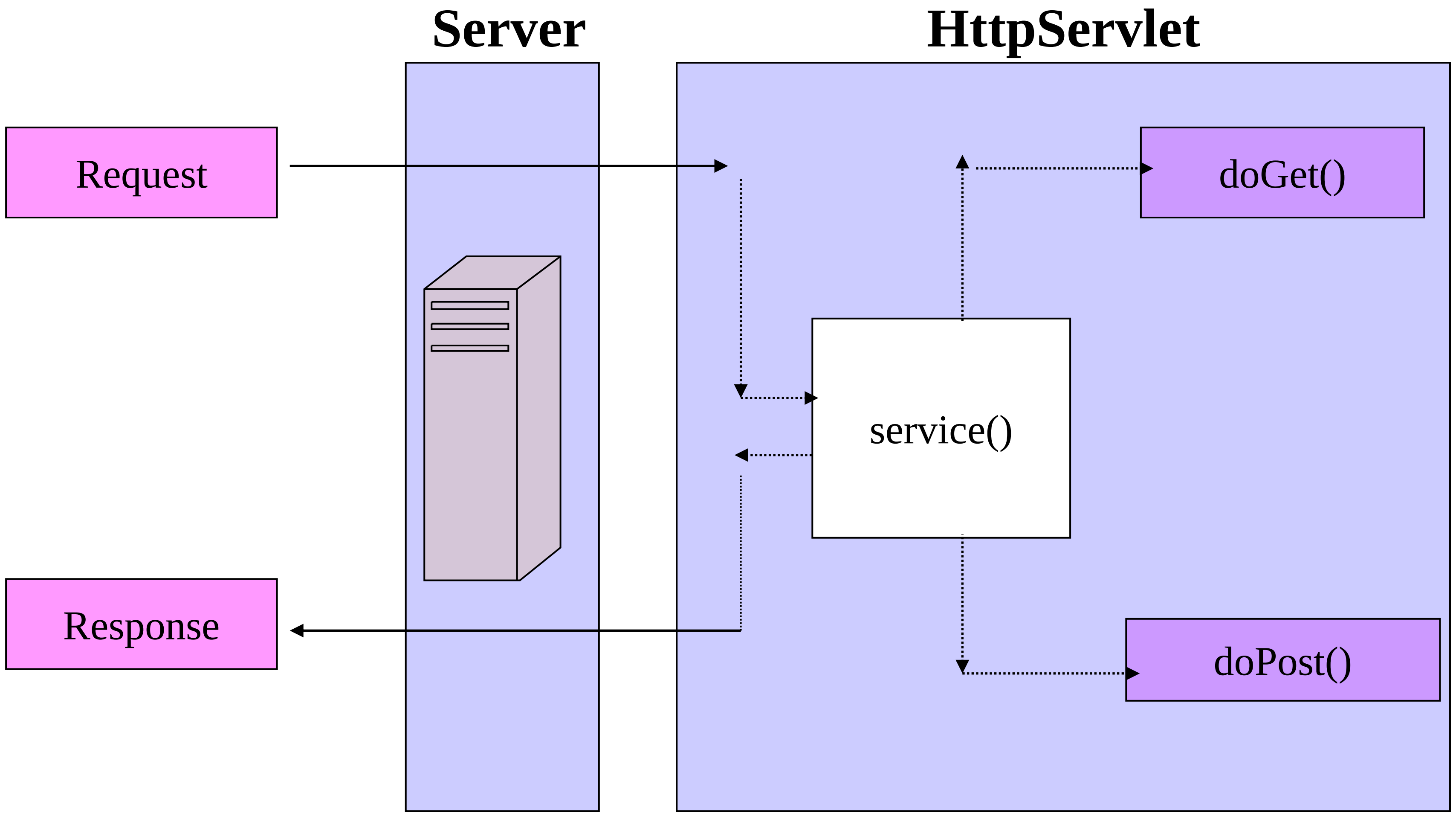
The HttpServlet Class

– It extends from jakarta.servlet.GenericServlet

– It contains extra methods like :

• doPost()	doPut()
• doGet ()	doDelete()
• doOptions()	doTrace()

The HttpServlet Class (cont'd)



HttpServlet Example

- The following code demonstrates the use of the **`doGet(..)`** method:

```
public void doGet(HttpServletRequest request,
                                HttpServletResponse response)
                                         throws ServletException, IOException
 {
     response.setContentType(“text/html”);
     PrintWriter out = response.getWriter();
     out.println("<HTML>");     out.println("<BODY>");
     out.println("The servlet has received a GET request." +
     "Now, click on the button below.");
     out.println("<BR>");
     out.println("<FORM METHOD=POST>");
     out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
     out.println("</FORM>");
     out.println("</BODY>");    out.println("</HTML>");
 }
```

HttpServlet Example cont'd

- The following code demonstrates the use of the **`doPost(..)`** method:

```
public void doPost(HttpServletRequest request,
                             HttpServletResponse response)
                                 throws ServletException, IOException
 {
     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<HTML>");     out.println("<HEAD>");
     out.println("<TITLE>The POST method</TITLE>");
     out.println("</HEAD>");
     out.println("<BODY>");
     out.println("The servlet has received a POST request.
                                               Thank you.");
     out.println("</BODY>");
     out.println("</HTML>");
 }
```

HttpServletRequest Interface

- Common methods:
  - `Enumeration getHeaderNames()`
  - `String getHeader(String headerName)`
  - `Enumeration getHeaders(String headerName)`
  - `String getQueryString()`
  - `HttpSession getSession(boolean create)`
  - `Cookie[] getCookies()`

- Remember, methods inherited from `ServletRequest`:
  - `Enumeration getParameterNames()`
  - `String getParameter(String paramName)`
  - `String[] getParameterValues(String paramName)`

HttpServletResponse Interface

- Common methods:
  - `setStatus(int statusCode)`
  - `sendRedirect(String newLocation)`
  - `sendError(int statusCode, String msg)`
  - `addCookie(Cookie cookie)`
  - `setHeader(String headerName, String value)`
  - `setIntHeader(String headerName, int value)`
  - `setDateHeader(String headerName, long milliseconds)`
- Remember, methods inherited from **`ServletResponse`**:
  - `PrintWriter getWriter()`
  - `setContentType(String mimeType)`
  - `setContentLength(int len)`

Status Code

- You have to set the status code and response headers before writing any document content to the response that will be sent back to the client.

- For the parameter of **`setStatus(..)`** method, you can either use explicit numbers of your own, or use the static constant values defined in the **`HttpServletResponse`** interface (e.g. for status code 404 use **`SC_NOT_FOUND`**).

The sendRedirect Method

- The **`sendRedirect(..)`** method is used to redirect the browser's client to go to another page other than the one originally requested.

- This is useful in several situations, for example:
  - If the requested page has been migrated to another server or url.
  - Deciding which page the user will be transferred to after successful login.
  - Checking if the browser enables the use of cookies.

- The location parameter can either be the relative URL of a page or its fully qualified URL.

The sendRedirect Method cont'd

- The **`sendRedirect(..)`** method makes a round trip to the client and then back to the server.

- A brand new request object is created, and the previous one is lost.

- Any data written to the response before calling it will be lost.

- If the calling servlet wishes to maintain some information from the old request and propagate it to the new request, then you can pass the information as a query string appended to the destination URL:

```
response.sendRedirect("MySecondServlet?userName="
                  + userName + "&password=" + password);
```

sendRedirect Example

- The following code demonstrates the use of **`sendRedirect(..)`** method:

```
public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                                throws ServletException, IOException
{
    .
    .
    out.println(“<FORM method= POST>”);
    out.println("<BR> Username: <INPUT TYPE=TEXT NAME=userName>");
    out.println("<BR> Password: <INPUT TYPE=PASSWORD
                                                NAME=password>");
    out.println("<BR> <INPUT TYPE=SUBMIT VALUE=Submit>");
    out.println("</FORM>");
    .
    .
}
```

sendRedirect Example cont’d

```
public void doPost(HttpServletRequest request,
                        HttpServletResponse response)
                             throws ServletException, IOException
{
    .
    .
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    If // check against Data Base
    {
      response.sendRedirect("WelcomeServletURL");
    }
    else { out.println(“Login Failed, please try again.”);
    // Draw the form again
    .}
}
```

Populating HTML Elements with Values

```
<INPUT TYPE=TEXT NAME=userName VALUE="ITI" >
```

- Explicit values example:

```
out.println("<INPUT TYPE=TEXT NAME=userName VALUE='ITI' >");
OR
 out.println("<INPUT TYPE=TEXT NAME=userName VALUE=\" ITI \" >");
```

- Using variables example:

```
out.println("<INPUT TYPE=TEXT NAME=userName VALUE=' " + userName + " ' >");
```

RequestDispatcher Interface

- The **`RequestDispatcher`** interface has 2 methods only:

  - **`include(ServletRequest request, ServletResponse response)`**

  - **`forward(ServletRequest request, ServletResponse response)`**

RequestDispatcher - include

- The **`include(..)`** method is used to include the output of another servlet or the content of another page (html, JSP, …etc) to the current servlet.

- This technique is famously used in websites to include a certain header or footer to all the pages of the website.

- It may also be used to include some common JavaScript functions to many web pages.

- The original servlet can write to the response before and after the included servlet or page writes to the response.

RequestDispatcher - forward

- The `forward(..)` method passes the processing of an HTTP request from the current servlet to another servlet or JSP (i.e. gives up the control for another servlet or jsp).
- It is up to the new servlet or JSP to write to the response object and then commit it to the client.
- Any output written to the response object by the original servlet will be lost (i.e. don’t write to the response before forwarding to another servlet).
- The original servlet can perform some initial tasks on the request object before forwarding it (e.g. put additional attributes on it, which will be used by the new servlet or jsp).

Using RequestDispatcher

- There are two ways to get a dispatcher for the desired destination servlet or page. Either from the request, or from the context.

- From the **`ServletRequest`**:
  - **`getRequestDispatcher(String path)`**

    //relative path or fully qualified path

- From the **`ServletContext`**:
  - **`getRequestDispatcher(String fullPath)`**

    // fully qualified path within the context, beginning with “/”

  - **`getNamedDispatcher(String servletName)`**

    // the servlet name in the web.xml file

Using RequestDispatcher - Example

- The following code demonstrates the use of the **`RequestDispather`**:

```
RequestDispatcher rd =
     request.getRequestDispatcher("SecondServletURL");

rd.include(request, response);
```

OR:

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

- Note: If you use **`include`** or **`forward`** from a **`doPost`** method, then the **`doPost`** method of the second servlet will be invoked.
- If you use them from a **`doGet`** method, then the **`doGet`** method of the second servlet will be invoked.
- Therefore, it is advisable to implement both **`doGet`** and **`doPost`** in the destination servlet (write the code in one of the two methods, and call it from the other method).

Page Transfer through RequestDispatcher

- **Include headers and footers to your page**

```
PrintWriter out = response.getWriter();
// add Header
out.println(“This the Header");
// real processing
request.setAttribute("username",“ITI");
RequestDispatcher rd =
    request.getRequestDispatcher("DispatcherServlet2");
rd.include(request, response);
// back to add footer
out.println(“This the Footer");
```

sendRedirect VS forward

- `sendRedirect(..)`:
  - goes through a round trip to the client side then back to the server.
  - The client’s original request will be lost.
  - To pass information, append a query string to the new location’s URL.

- `forward(..)`:
  - Redirects the request internally inside the server.
  - Both the request and the response are passed to the new resource.

Lab Exercise

Submit User Details to a Servlet

- **<u>Login :</u>**
  - Write a Servlet for Login page that has two text input fields for the user’s name and mail, along with a submit button (use doGet(..)).
  - Upon pressing the submit button, the page should send the data to a servlet(the doPost(..) of the same Servlet of the Login).
  - Upon submitting the login form, the user should be redirected to a welcome page(in case of successful login.. Try to send user name to it).
  - If the login data is incorrect, display a colored error message on the login page.

Forwarding to Another Servlet

- Update Login servlet to do checks on the Parameters(not NULL) and then forward to another servlet to display the pervious request parameters and attributes.

- Try to make the first servlet write to the response before forwarding it to the second servlet, and then see what happens.

Including Header and Footer

- Create header page (menu page)and a footer page and then include them in your servlet (at the beginning and end).

Chapter 5

Accessing Databases with JDBC

Chapter 4 Outline

- ❑ Review on JDBC
- ❑ Steps to Work with a Database
- ❑ Sample Applications of Database Operations
- ❑ Online SQL Tool

Review on JDBC

- **JDBC API: s**et of Java interfaces for connecting to any DBMS and executing SQL statements. Packages are:
  - java.sql package: basic functionalities.
  - javax.sql package: advanced functionalities.

- **JDBC Driver:** Vendor-specific implementation of JDBC interfaces. Every DBMS has JDBC driver(s).

- **Connection String:** A special DBMS-specific string that is used to connect to the database, for example:
  - **MySQL:**
    "jdbc:mysql://localhost:3306/mysql", "username", "pass"
  - **Oracle (thin):**
    "jdbc:oracle:thin:@localhost:1521:xe", "username", "pass"

Steps to Work with a Database

1. Load (register) the JDBC driver.
2. Obtain a connection.
3. Execute query statements.
4. Process results if any (**`ResultSet`**).

Steps to Work with a Database cont'd

- First Step:
  - **Put the driver (if jar file) in WEB-INF/lib for deploying** (and in CLASSPATH environment variable for development).
  - The driver could also be put in the common/lib folder (will be common among all web applications).
  - Register an object of the driver, for example:

```
DriverManager.registerDriver(
        new oracle.jdbc.driver.OracleDriver());
```

- Second Step:
  - Obtain the connection through the Connection interface, for example:

```
Connection con = DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe", "username", "pass");
```

Steps to Work with a Database cont'd

- Third Step:
  - Create a statement and execute queries using either **`executeUpdate(..)`** or **`executeQuery(..)`**, for example:

```
Statement s = connection.createStatement();
ResultSet rs= s.executeQuery("select * from Emp");
```

- Fourth Step:
  - After obtaining the **`ResultSet`**, extract data through the getter methods, for example:

```
while(rs.next())
{
   out.println(rs.getInt(1) + " : " + rs.getString( "FNAME"
   ));
}
```

- Note: You can get information about the retrieved **`ResultSet`** by using the method: **`getMetaData()`**

Steps to Work with a Database cont’d

- Do not forget to close the connection.
- By default, only one **`ResultSet`** object per Statement object can be open at the same time.
- If the default **`ResultSet`** object is obtained, then columns within each row should be read in left-to-right order, and each column should be read only once.

Steps to Work with a Database cont’d

- You can also Define DataSource in the context.xml file

- In your web application folder make a folder named
  META-INF inside this folder put context.xml file

In the context.xml put the following element

```
<Context>
    <Resource name= "jdbc/TestDB" auth="Container" type= "javax.sql.DataSource"
            maxTotal="100" maxIdle="30" maxWaitMillis="10000"
            username="root" password="root" driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://localhost:3306/test"/>
</Context>
```

Steps to Work with a Database cont'd

- In your web.xml

```
<resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/TestDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

- In your code

```
Context initContext = new InitialContext();
Context envContext  = (Context)initContext.lookup("java:/comp/env");
DataSource ds = (DataSource)envContext.lookup("jdbc/TestDB");
Connection conn = ds.getConnection();
….etc.
```

Lab Exercise

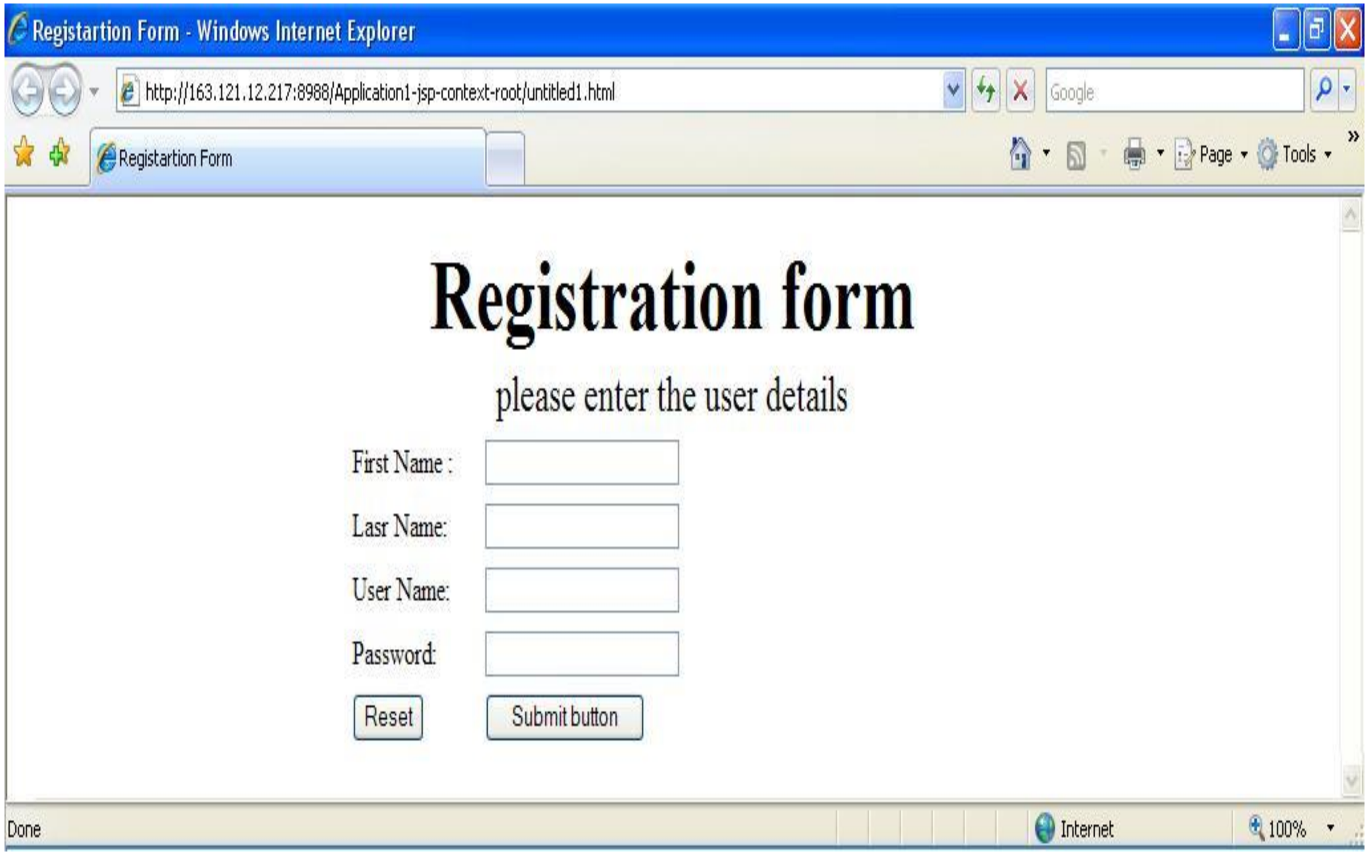
Login Page

– Update the Login Servlet to check for the userName and Password against the Database

Inserting Records into the Database

- You can insert records using the following SQL statement:

```
insert into Users (FirstName, LastName, UserName, Password)
                    values('JETS', 'ITI', 'jets', 'iti')
```



Registartion Form - Windows Internet Explorer

http://163.121.12.217:8988/Application1-jsp-context-root/untitled1.html

Registration form

please enter the user details

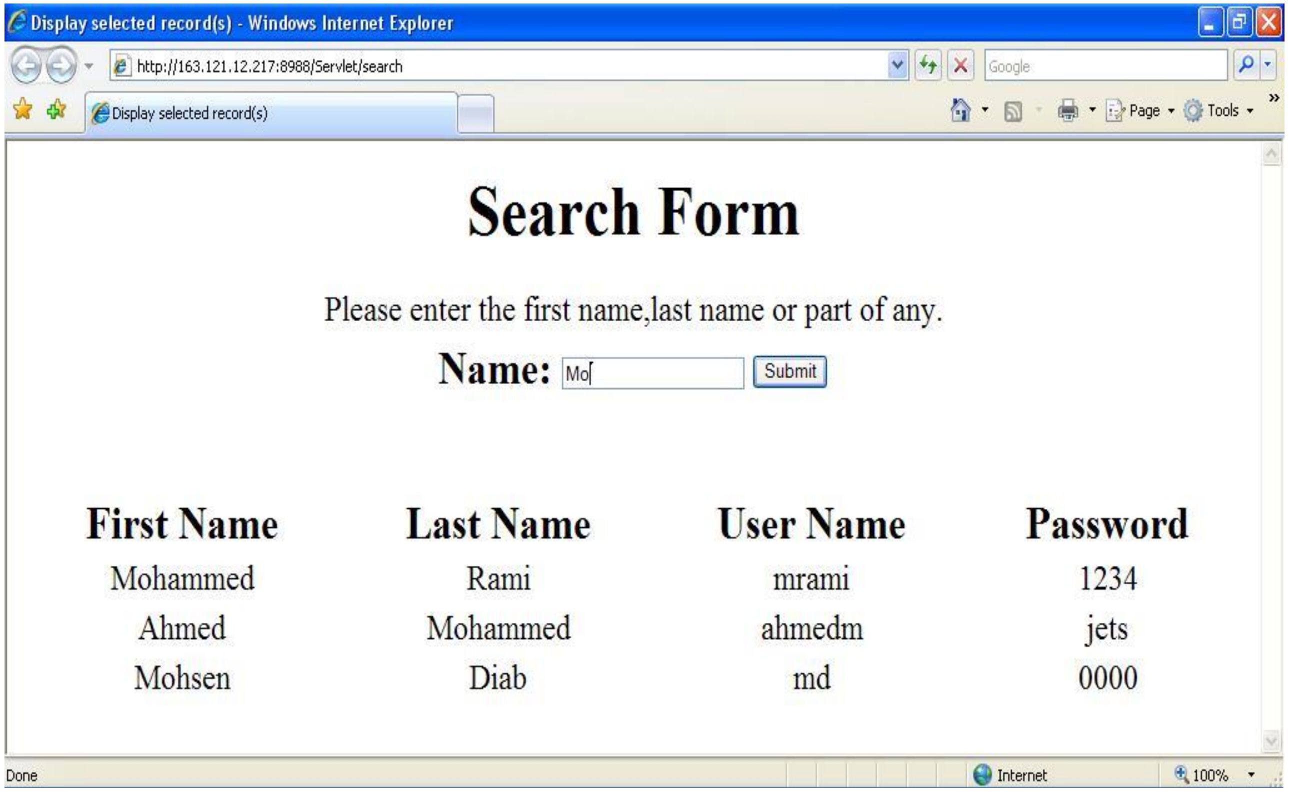
First Name :	
Lasr Name:	
User Name:	
Password:	
Reset	Submit button

Done

Searching for Records in a Database

• You can write an SQL statement that searches for a certain name in the database:

select FirstName, LastName, UserName, Password FROM Users WHERE FirstName LIKE '%*keyword*%' OR LastName LIKE '%*keyword*%'



Display selected record(s) - Windows Internet Explorer
http://163.121.12.217:8988/Servlet/search
Google
Display selected record(s)
Page
Tools
Done
Internet
100%

Search Form

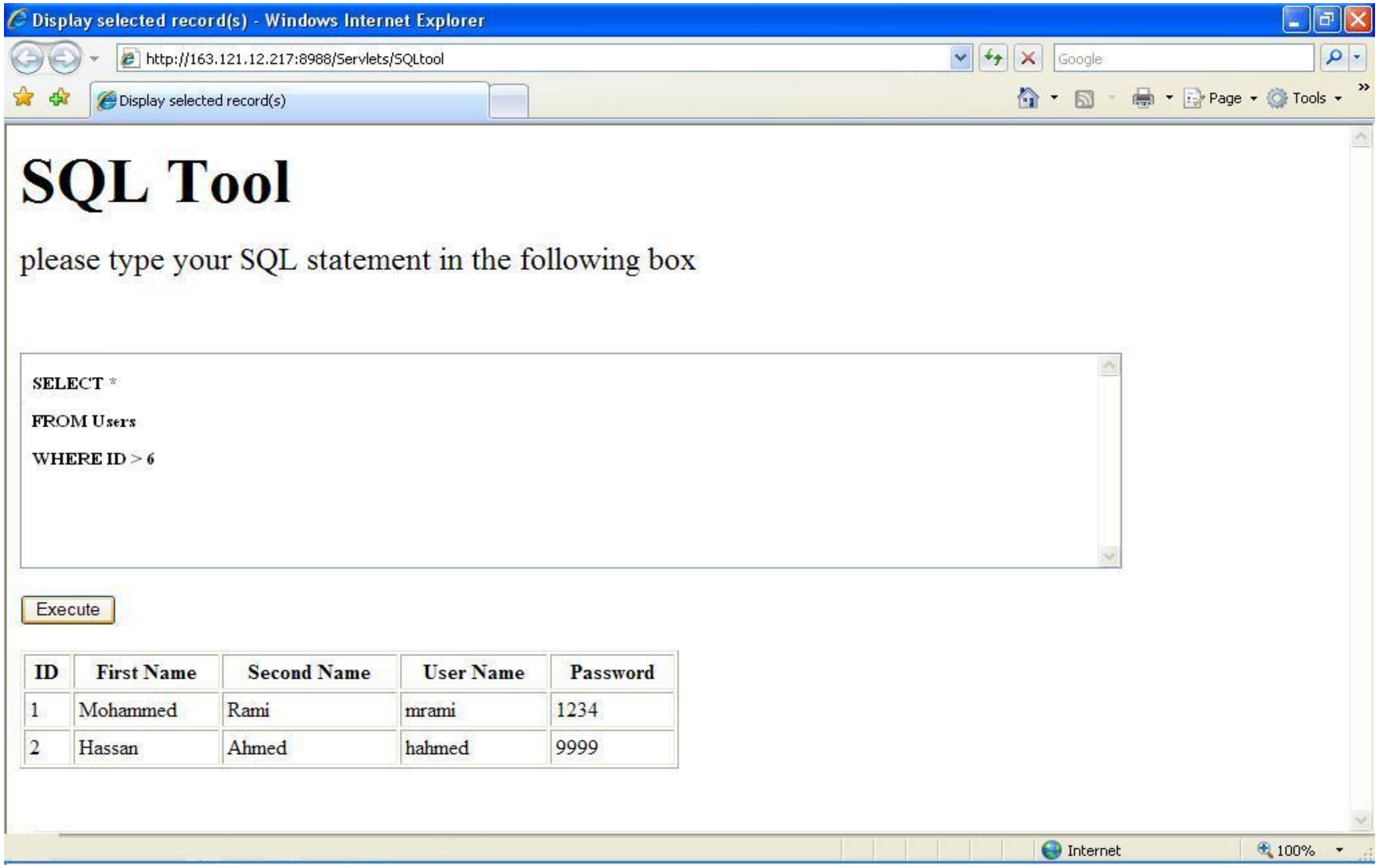
Please enter the first name,last name or part of any.

Name: Mo Submit

First Name	Last Name	User Name	Password
Mohammed	Rami	mrami	1234
Ahmed	Mohammed	ahmedm	jets
Mohsen	Diab	md	0000

Online SQL Tool

• You can now create an online SQL tool that executes any query on the database and display the result in a text area.


Display selected record(s) - Windows Internet Explorer
http://163.121.12.217:8988/Servlets/SQLtool
Google
Display selected record(s)
Page
Tools
Internet
100%

SQL Tool

please type your SQL statement in the following box

```
SELECT *
FROM Users
WHERE ID > 6
```

Execute

ID	First Name	Second Name	User Name	Password
1	Mohammed	Rami	mrami	1234
2	Hassan	Ahmed	hahmed	9999

Chapter 6

Session Management

Chapter 6 Outline

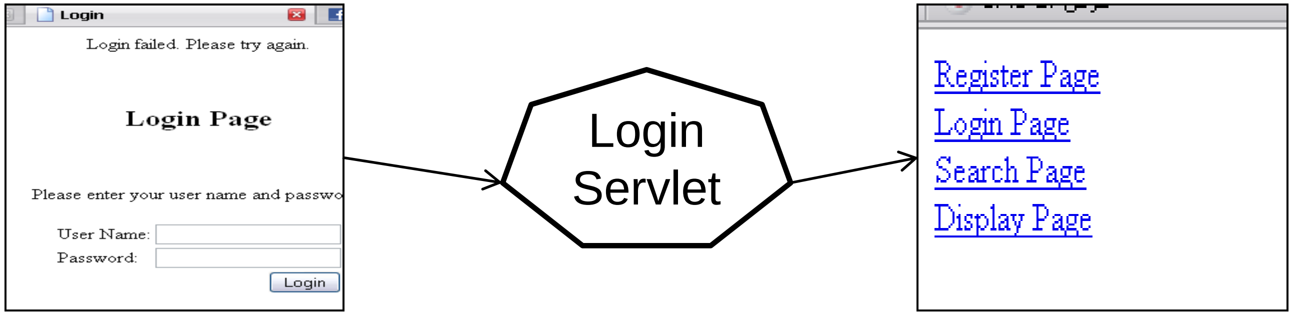
- ❑ Session Management Techniques
- ❑ URL Rewriting
- ❑ Hidden Fields
- ❑ Cookies
- ❑ Session Objects

Session Management

- Session management is the ability to keep track of (maintain) the user’s state through several web pages.
- Remember: the problem with the HTTP protocol is that it is a stateless protocol.
- Session management is very useful for the following:
  - Identifying (remembering) the user during ecommerce sessions.
  - Remembering usernames and passwords throughout multiple pages.
  - Customizing website’s view according to user’s preferences.
  - Focused advertising.
- The main idea of maintaining a session is to send a certain token (identifier) with the response.

Session Management Techniques

- The following techniques can be used separately or cooperatively in order to maintain the client's information throughout the entire session:
  - URL rewriting.
  - Hidden fields.
  - Cookies.
  - Session objects.


Login

Login failed. Please try again.

Login Page

Please enter your user name and passwo

User Name:

Password:

Login

URL Rewriting

- URL rewriting is simply appending token(s) to the URL of the next page or servlet. It starts by a question mark
- Tokens are strings in the form of parameter=value
- Tokens are separated by an ampersand (&)
- It is an easy way but has the following concerns:
  - Supports up to 2000 characters only.
  - The parameters are exposed in the URL.
  - You will need to encode special characters (e.g. spaces to + and special characters to %XX).
- Example:

  http://www.site.com/mySerevlet?param1=value1&param2=value2

Hidden Fields

- Forms can include hidden fields among the normal input fields.
- Therefore, information can be stored in hidden fields and then retrieved from the request in the next page or servlet.
- The concern is that the user can view the values of the hidden fields by viewing the html source of the page from within the browser.
- Example:

```
out.print("<INPUT TYPE=HIDDEN Name=id VALUE=" +myId+ ">");
```

Cookies

- A cookie is a small piece of information (name and value) that is passed back and forth in the HTTP request and response.

- The cookie sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

- Cookies are represented as a class in the HTTP package.

- To create a cookie you write such code:

```
Cookie c1 = new Cookie("myCookieName", “myValue");
```

- Niether the name nor the value can contain commas, semicolons, white space, nor begin with a $ character.

Cookies cont'd

- To add a cookie to the response:

```
Cookie c1 = new Cookie("myCookieName", “myValue");
response.addCookie(c1);
```

- To retrieve a cookie from the request:

```
Cookie[] cookies = request.getCookies();
if(cookies != null)
{
   for (int i=0; i<cookies.length; i++)
   {
      Cookie cookie = cookies[i];
      out.println("<B>Cookie Name:</B> " +
                                   cookie.getName() + "<BR>");
      out.println("<B>Cookie Value:</B> " +
                                  cookie.getValue() + "<BR>");
   }
}
```

Cookies cont’d

- Cookies must be added to the response before writing any other response elements.

- Advantages of cookies:
  - The values of the cookies are not directly visible.
  - You don't need to use a form tag.
  - Cookies are not a serious security threat.

- Concerns about cookies:
  - The user has the choice to set the browser to refuse cookies.
  - Privacy: if another user has access to the PC, then he may be able to view the values of stored cookies on the hard disk. Therefore, make sure not to write critical information in the cookie. (Anyway, there’s no security without physical security).

Cookies cont'd

- You can detect whether a browser enables cookies or not, and display a message that asks the user to enable cookies.

- To detect this, you can do the scenario of adding a cookie to the response of the first servlet, then redirecting to a second servlet that checks the existence of the cookie in the request.

- To redirect the client's browser to another page you can use either of the following:
  - **`response.sendRedirect(..)`**
  - <META **HTTP-EQUIV**=Refresh **CONTENT**="time to wait **;URL**=url">

- Note: in both cases you have to use a relative URL, not a fully qualified one.

Cookies cont’d

- A cookie can either be stored on session level or on persistence level.
- A session level cookie is a cookie that is stored in the browser’s memory and deleted when the user quits the browser.
- A persistent cookie is stored on the client’s hard disk. To make your cookie persistent, use the method **`setMaxAge(int seconds)`** before adding it to the response.
- Example: **`c1.setMaxAge(60*60*24*7);`** // One week

Lab Exercise

1. Testing if browser enables cookies

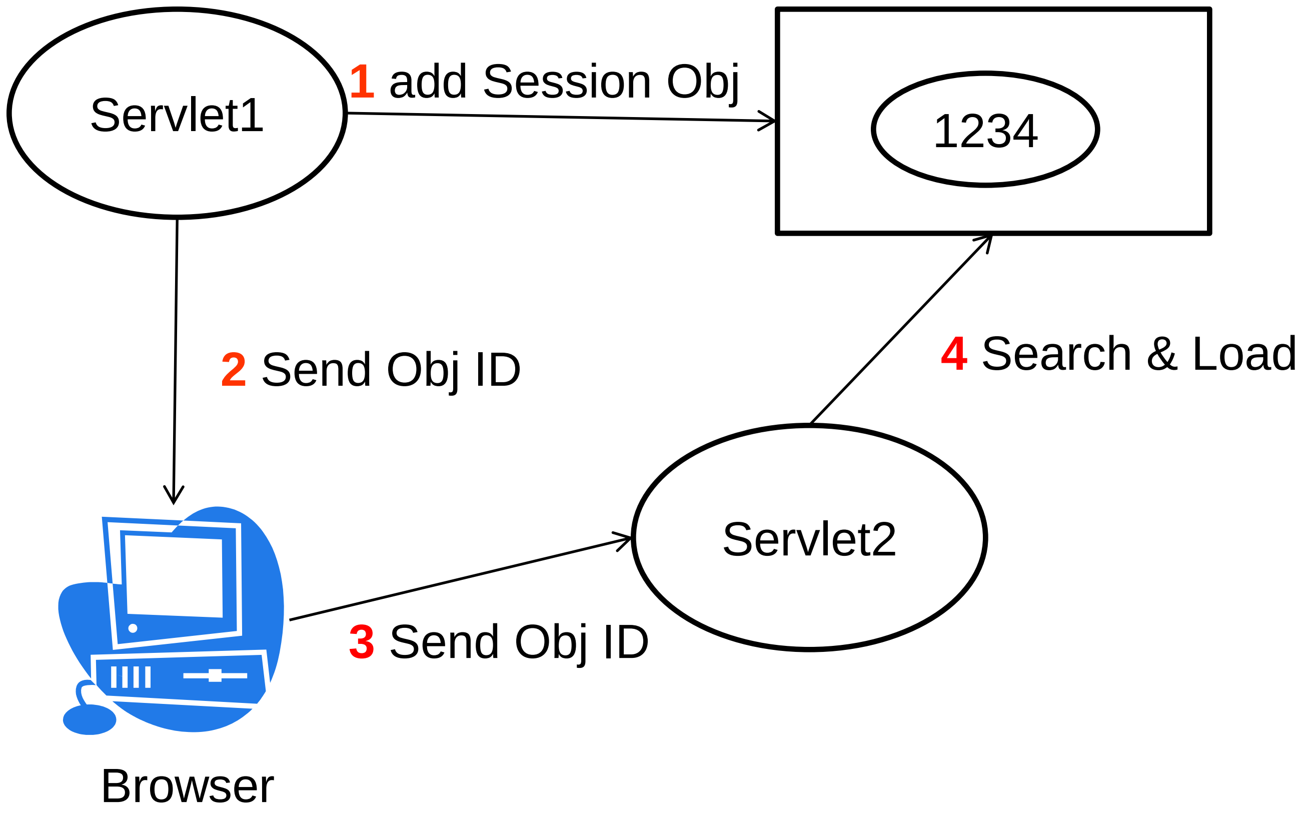
- Implement the cookie testing scenario by developing a servlet that adds a cookie to a response and then redirecting to another servlet that attempts to retrieve the cookie.

- If the user has disabled the use of cookies, send him an error message asking him to enable cookies.

Session Objects

- The best and most powerful was is to use session objects (defined by the **`HTTPSession`** interface).
- For each user, the servlet container can create an **`HttpSession`** object that is associated with that user only.
- A session objects act like a Hashtable into which you can store any number of key/object pairs.
- It is accessible from other servlets or jsp pages in the same application (per user).

Session Objects Basic Scenario



Session Objects Basic Scenario cont'd

- The basic scenario of creating and then using a session object is as follows:

  1. A session object is created by Servlet1 (using the **`getSession(..)`** method), where a unique session identifier is generated for it (e.g. jsessionid=1234).

  2. Servlet1 implicitly sends the session identifier to the client browser.

  3. When the client browser requests another resource from the same application, such as Servlet2, the session identifier is implicitly sent back to the server and passed to Servlet2 in the request object.

  4. For Servlet2 to access the session object of this particular client, it uses the **`getSession(..)`** method which automatically retrieves the session identifier from the request and gains access to the specific session object associated with that session identifier.

Session Objects - Example

- In `doPost(..)` method of the LoginServlet:

```
// After user enters correct login information
HttpSession session = request.getSession(true);
session.setAttribute("loggedIn", new String("true"));
response.sendRedirect("ContentServletURL");
```

- In `doGet(..)` method of the ContentServlet:

```
HttpSession session = request.getSession(false);
if (session == null)
  response.sendRedirect(“LoginServletUrl”);
else
{
  String loggedIn = (String) session.getAttribute("loggedIn");
  if (!loggedIn.equals("true"))
     response.sendRedirect (“loginServletUrl”);
}
```

HttpSession

- Common methods:
  - `Enumeration getAttributeNames()`
  - `Object getAttribute(String key)`
  - `void setAttribute(String key, Object value)`
  - `void removeAttribute (String key)` // e.g. log out
  - `long getCreationTime()`
  - `String getId()`
  - `long getLastAccessedTime()`
  - `void setMaxInactiveInterval(int seconds)`
  - `int getMaxInactiveInterval()`
  - `ServletContext getServletContext()`
  - `void invalidate()`
  - `boolean isNew()`

Invalidating a Session

- A session can become invalid in 3 cases:
  - The default session timeout has elapsed (configured in the servlet container for all web applications).
  - The maximum inactive interval time of the session has elapsed (specified by calling the method `setMaxInactiveInterval(..)`).
  - The programmer decides to invalidate a session (by calling the `invalidate()` method).

Lab Exercise

2. User login

- Keep user ‘s login information on the session and access it in another page

Chapter 7

Servlet Annotation

@WebServlet annoation

- Starting from JEE 6
- Package jakarta.servlet.annotation

```
@WebServlet(
   attribute1=value1,
   attribute2=value2,
   ...
)
public class TheServlet extends jakarta.servlet.http.HttpServlet
{
   // servlet code...

}
```

@WebServlet annotation (cont.)

Name	Type	Required	Description
value or **urlPatterns**	*String[]*	Required	Specify one or more URL patterns of the servlet. Either of attribute can be used, but not both.
name	*String*	Optional	Name of the servlet
displayName	*String*	Optional	Display name of the servlet
description	*String*	Optional	Description of the servlet
asyncSupported	*boolean*	Optional	Specify whether the servlet supports asynchronous operation mode. Default is false.
initParams	*WebInitParam[]*	Optional	Specify one or more initialization parameters of the servlet. Each parameter is specified by `@WebInitParam` annotation type.
loadOnStartup	*int*	Optional	Specify load-on-startup order of the servlet.
smallIcon	*String*	Optional	Specify name of the small icon of the servlet.
largeIcon	*String*	Optional	Specify name of the large icon of the servlet.

@WebServlet annotation (cont.)

```
@WebServlet("/processForm")
public class TheServlet extends jakarta.servlet.http.HttpServlet {
  // servlet code...
}


@WebServlet(urlPatterns = {"/sendFile", "/uploadFile"})
public class TheServlet extends jakarta.servlet.http.HttpServlet {
  // servlet code...
}


 @WebServlet(
     name = "MyOwnServlet",
     description = "This is my first annotated servlet",
     urlPatterns = "/processServlet"
 )
```

@WebServlet annotation (cont.)

```
@WebServlet(
    urlPatterns = "/imageUpload",
    initParams =
    {
      @WebInitParam(name = "saveDir", value = "D:/FileUpload"),
      @WebInitParam(name = "allowedTypes", value = "jpg,jpeg,gif,png")
    }
)
```

```
@WebServlet(
    urlPatterns = "/myController",
    loadOnStartup = 1,
    asyncSupported = true
)
```

References & Recommended Reading

- Core Servlets and JSP.
- Java for the Web with Servlets, JSP, and EJB.
- Manning JSTL in Action.
- Sun presentations.
- Oracle presentations.
- SCJWD study guide.