

UNIT-V

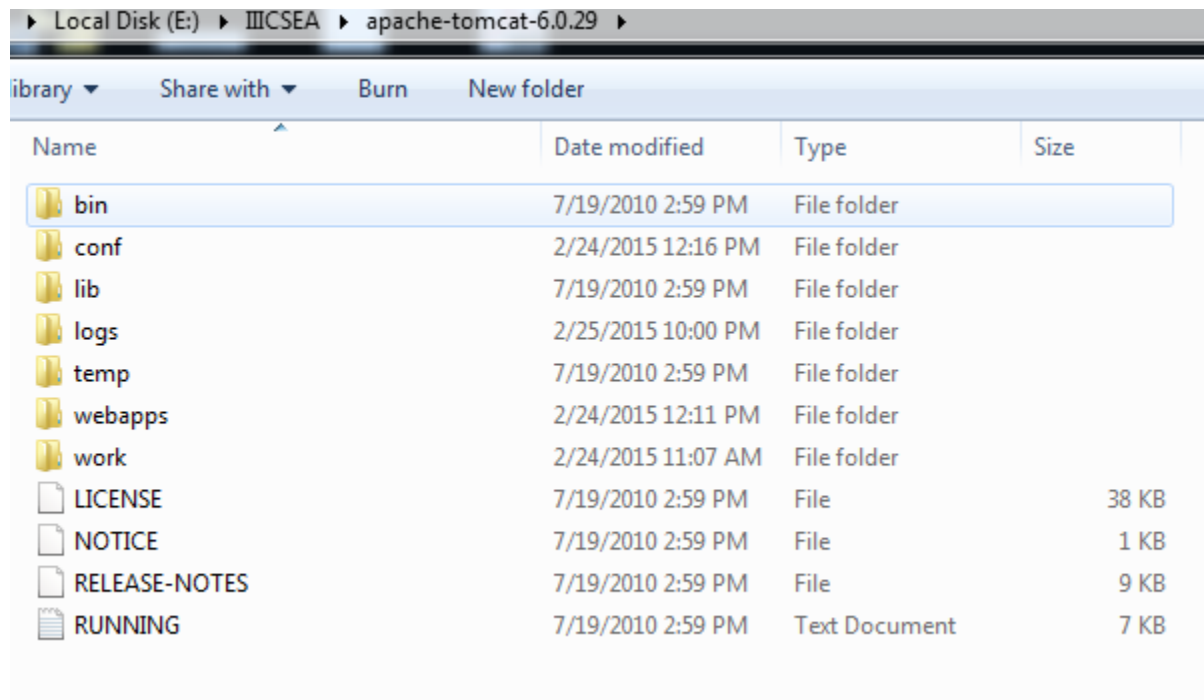
Lecture-1

Tomcat Web Server

Apache Tomcat Servlet/JSP container. Apache Tomcat version 6.0 implements the Servlet 2.5 and JavaServer Pages 2.1 specifications from the Java Community Process, and includes many additional features that make it a useful platform for developing and deploying web applications and web services.

These are some of the key tomcat directories:

- **/bin** - Startup, shutdown, and other scripts. The *.sh files (for Unix systems) are functional duplicates of the *.bat files (for Windows systems). Since the Win32 command-line lacks certain functionality, there are some additional files in here.
- **/conf** - Configuration files and related DTDs. The most important file in here is server.xml. It is the main configuration file for the container.
- **/logs** - Log files are here by default.
- **/webapps** - This is where your webapps go.



Installing Tomcat on Windows can be done easily using the Windows installer. Its interface and functionality is similar to other wizard based installers, with only a few items of interest.

- **Installation as a service:** Tomcat will be installed as a Windows service no matter what setting is selected. Using the checkbox on the component page sets the service as "auto" startup, so that Tomcat is automatically started when Windows starts. For optimal

security, the service should be run as a separate user, with reduced permissions (see the Windows Services administration tool and its documentation).

- **Java location:** The installer will provide a default JRE to use to run the service. The installer uses the registry to determine the base path of a Java 5 or later JRE, including the JRE installed as part of the full JDK. When running on a 64-bit operating system, the installer will first look for a 64-bit JRE and only look for a 32-bit JRE if a 64-bit JRE is not found. It is not mandatory to use the default JRE detected by the installer. Any installed Java 5 or later JRE (32-bit or 64-bit) may be used.
- **Tray icon:** When Tomcat is run as a service, there will not be any tray icon present when Tomcat is running. Note that when choosing to run Tomcat at the end of installation, the tray icon will be used even if Tomcat was installed as a service.

Installation:

JDK:

Tomcat 6.0 was designed to run on J2SE 5.0.

Tomcat:

Binary downloads of the **Tomcat** server are available from <http://tomcat.apache.org/download-60.cgi>. This manual assumes you are using the most recent release of Tomcat 6.

Web Application Directory Structure

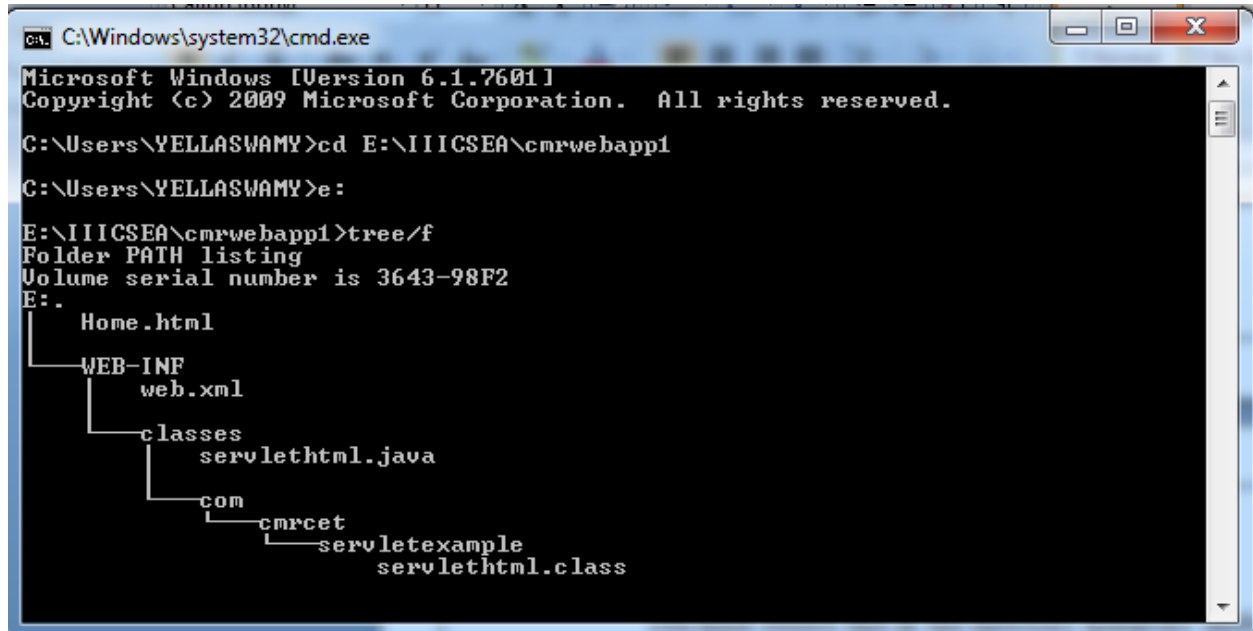
To facilitate creation of a Web Application Archive file in the required format, it is convenient to arrange the "executable" files of your web application (that is, the files that Tomcat actually uses when executing your app) in the same organization as required by the WAR format itself. To do this, you will end up with the following contents in your application's "document root" directory:

- ***.html, *.jsp, etc.** - The HTML and JSP pages, along with other files that must be visible to the client browser (such as JavaScript, stylesheet files, and images) for your application. In larger applications you may choose to divide these files into a subdirectory hierarchy, but for smaller apps, it is generally much simpler to maintain only a single directory for these files.
- **/WEB-INF/web.xml** - The *Web Application Deployment Descriptor* for your application. This is an XML file describing the servlets and other components that make up your application, along with any initialization parameters and container-managed security constraints that you want the server to enforce for you. This file is discussed in more detail in the following subsection.
- **/WEB-INF/classes/** - This directory contains any Java class files (and associated resources) required for your application, including both servlet and non-servlet classes, that are not combined into JAR files. If your classes are organized into Java packages,

you must reflect this in the directory hierarchy under `/WEB-INF/classes/`. For example, a Java class named `com.mycompany.mypackage.MyServlet` would need to be stored in a file named `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class`.

- **/WEB-INF/lib/** - This directory contains JAR files that contain Java class files (and associated resources) required for your application, such as third party class libraries or JDBC drivers.

Example:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\YELLASWAMY>cd E:\IIICSEA\cmrwebapp1
C:\Users\YELLASWAMY>e :
E:\IIICSEA\cmrwebapp1>tree /f
Folder PATH listing
Volume serial number is 3643-98F2
E:
├── Home.html
├── WEB-INF
│   ├── web.xml
│   └── classes
│       ├── servlethtml.java
│       └── com
│           └── cmrcet
│               └── servletexample
│                   └── servlethtml.class
```

Web Application Deployment Descriptor:

As mentioned above, the `/WEB-INF/web.xml` file contains the Web Application Deployment Descriptor for your application. As the filename extension implies, this file is an XML document, and defines everything about your application that a server needs to know (except the *context path*, which is assigned by the system administrator when the application is deployed).

Example:

Web.xml

```
1 <web-app>
2 <servlet>
3 <servlet-name>servlethtml</servlet-name>
4 <servlet-class>com.cmrcet.servletexample.servlethtml.class</servlet-class>
5 </servlet>
6 <servlet-mapping>
7 <servlet-name>servlethtml</servlet-name>
8 <url-pattern>/getHomeHtml</url-pattern>
9 </servlet-mapping>
10 </web-app>
```

Deployment with Tomcat 6

In order to be executed, a web application must be deployed on a servlet container. This is true even during development. We will describe using Tomcat 6 to provide the execution environment. A web application can be deployed in Tomcat by one of the following approaches:

- *Copy unpacked directory hierarchy into a subdirectory in directory* `$CATALINA_BASE/webapps/`. Tomcat will assign a context path to your application based on the subdirectory name you choose. We will use this technique in the `build.xml` file that we construct, because it is the quickest and easiest approach during development. Be sure to restart Tomcat after installing or updating your application.
- *Copy the web application archive file into directory* `$CATALINA_BASE/webapps/`. When Tomcat is started, it will automatically expand the web application archive file into its unpacked form, and execute the application that way. This approach would typically be used to install an additional application, provided by a third party vendor or by your internal development staff, into an existing Tomcat installation. **NOTE** - If you use this approach, and wish to update your application later, you must both replace the web application archive file **AND** delete the expanded directory that Tomcat created, and then restart Tomcat, in order to reflect your changes.
- *Use the Tomcat 6 "Manager" web application to deploy and undeploy web applications.* Tomcat 6 includes a web application, deployed by default on context path `/manager`, that allows you to deploy and undeploy applications on a running Tomcat server without restarting it. See the administrator documentation (TODO: hyperlink) for more information on using the Manager web application.
- *Use "Manager" Ant Tasks In Your Build Script.* Tomcat 6 includes a set of custom task definitions for the `Ant` build tool that allow you to automate the execution of commands to the "Manager" web application. These tasks are used in the Tomcat deployer.

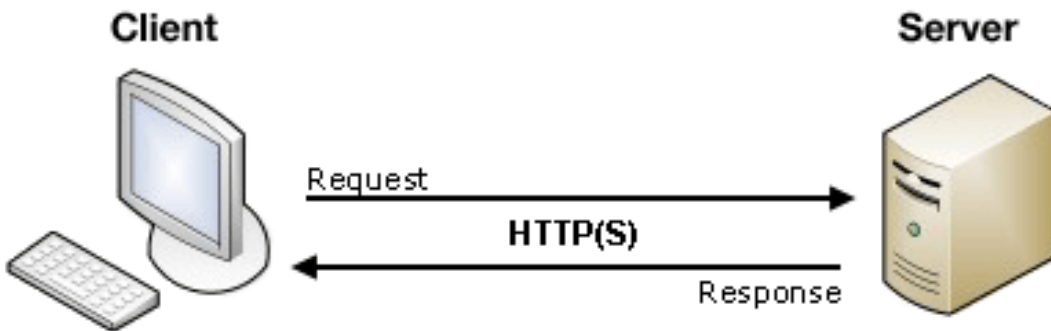
- *Use the Tomcat Deployer.* Tomcat 6 includes a packaged tool bundling the Ant tasks, and can be used to automatically precompile JSPs which are part of the web application before deployment to the server.

Deploying your app on other servlet containers will be specific to each container, but all containers compatible with the Servlet API Specification (version 2.2 or later) are required to accept a web application archive file. Note that other containers are **NOT** required to accept an unpacked directory structure (as Tomcat does), or to provide mechanisms for shared library files, but these features are commonly available.

UNIT-V

Lecture-2

Client Server Architecture



The web is based on a very basic client/server architecture that can be summarized as follows:

- A client (usually a Web browser) sends a request to a server (most of the time a web server like [Apache](#), [Nginx](#), [IIS](#), [Tomcat](#), etc.), using the [HTTP protocol](#).
- The server answers the request using the same protocol.

CERN=Charmed quarks stored at European organization for nuclear Research

MIME-Multipurpose Internet mail extensions

Protocols:

HTTP	:Hypertext Transfer Protocol
FTP	:File Transfer Protocol
TCP	:Transmission Control Protocol
IP	:Internet Protocol
UDP	:User Datagram Protocol
Telnet	

Difference between FTP and HTTP

FTP-Transfers files from server to client support several requests for connection

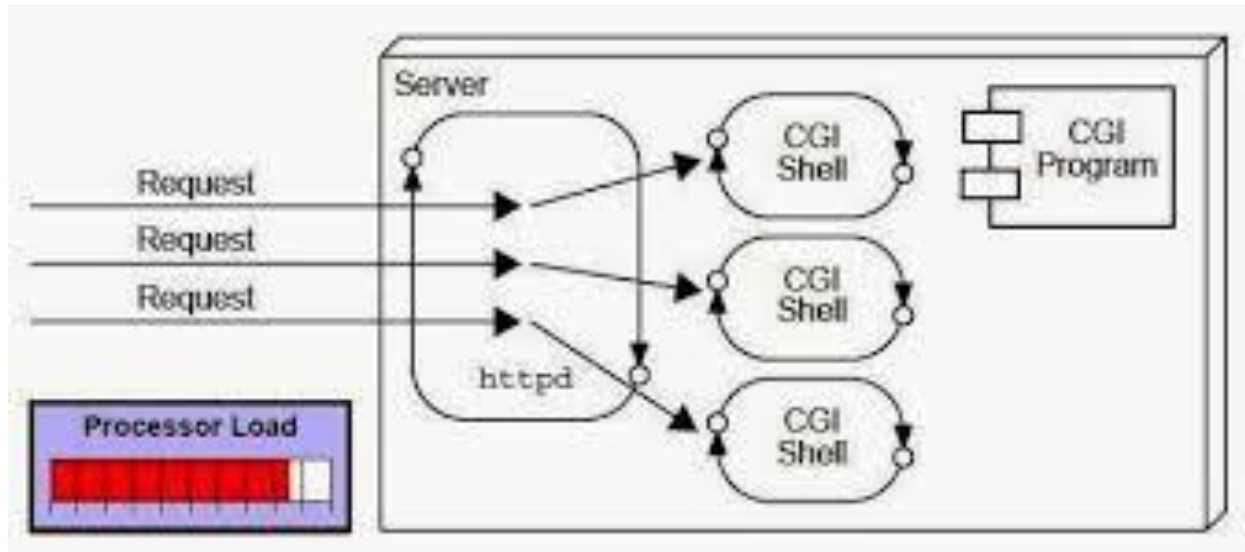
HTTP-support only one request for connection

The first web browser is Mosaic created by the National Center for Supercomputing Applications(NCSA)

Web Applications and Web Sites

Early in the development of HTML, the designers created a mechanism to permit a user to invoke a program on the web server. This mechanism was called the Common Gateway Interface (CGI). When a website includes CGI Processing this is called a web application.

CGI Programs on the Web Server



Execution of CGI Programs

At runtime a CGI program is launched by the web server as a separate operating system (OS) shell.

The shell includes an OS environment and process to execute the code of the CGI program, which resides within the server's file system.

Advantages of CGI Programs

1. A program can be written in a variety of languages although they are primarily written in Perl
2. A CGI Program with bugs does not crash the web server
3. Programs are easy for a web designer to reference. When the script is written, the designer can reference it in one line in a web page.
4. Because CGI Programs execute in their own OS Shell, these programs do not have concurrency conflicts with other HTTP requests executing the same CGI program.
5. All service providers support CGI Programs

Disadvantages of CGI Programs

1. The response time of CGI programs is high because CGI programs execute in their own OS Shell. The creation of OS shell is heavyweight activity for the OS
2. CGI is not Scalable. If the number of people accessing the web application increases from 50 to 5000, it can not be adopted to handle the load.
3. The languages for CGI are not always secure or object-oriented.
4. The CGI script has to generate an HTML response, so the CGI code is mingled with HTML. This is not good separation of presentation and business logic.
5. Scripting languages are often platform-dependent.

Note: To overcome above disadvantage Servlets are introduced.

Java Servlets:

Sun Microsystems developed servlets as an advance over traditional CGI technology

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extends the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Servlet Programs on the Web Server:

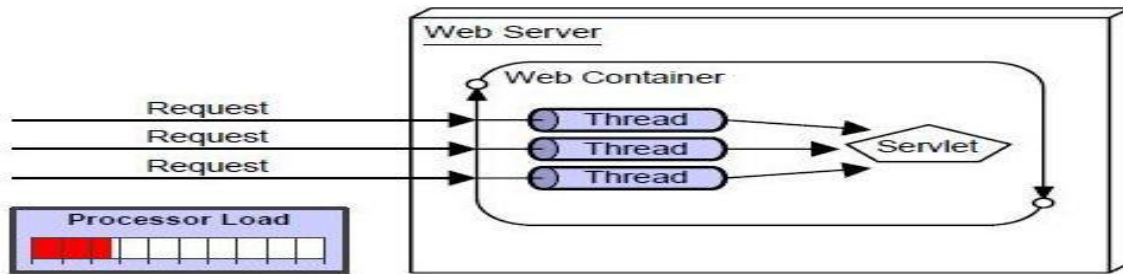
Unlike CGI programs, servlets run within component container architecture. This container is called web container or servlet engine.

- The web container is a JVM that supplies an implementation of the Servlet API
- Servlet instances are components that are managed by the web container to respond to HTTP requests.

Execution of Java Servlets

1. The basic processing steps for Java servlets are quite similar to the steps for CGI. However, the servlet runs as a thread in the web container instead of in a separate OS process.

2. When the number of requests for a servlet rises, no additional instances of the servlet or operating system process are created.
3. Each request is processed concurrently using one Java thread per request.



Advantages of Java Servlets:

1. Each request is run in a separate thread, so servlet request processing is significantly faster than traditional CGI Processing.
2. Servlets are Scalable. Many more requests can be executed because the web container uses a thread rather than an operating system process, which is a limited system resource.
3. Servlets are robust and object oriented.
4. Servlets are platform independent
5. The web container provides additional services to the servlets, such as error handling and security.

Disadvantages of Java Servlets:

1. Servlets often contain both **business logic** and **presentation logic**.
2. Servlets must handle concurrency issues
3. Mixing presentation and business logic means that whenever a webpage changes the servlets must be rewritten, recompiled and redeployed.

Presentation Logic: is anything that controls how the application presents information to the user.

Business Logic: is anything that manipulates data to accomplish something, such as storing data.

Note: To overcome above servlet disadvantages Java Server Pages (JSP) are introduced.

UNIT-V

Lecture-3

The Servlet Container

It provides runtime environment for JavaEE (j2ee) applications.

It performs many operations that are given below:

1. Life Cycle Management
2. Multithreaded support
3. Object Pooling
4. Security etc.

Server

It is a running program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

Web Server

Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Example of Web Servers are: **Apache Tomcat** and **Resin**.

Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Example of Application Servers are:

1. **JBoss** Open-source server from JBoss community.
2. **Glassfish** provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic** provided by Oracle. It more secured.

4. **Websphere** provided by IBM.

Content Type

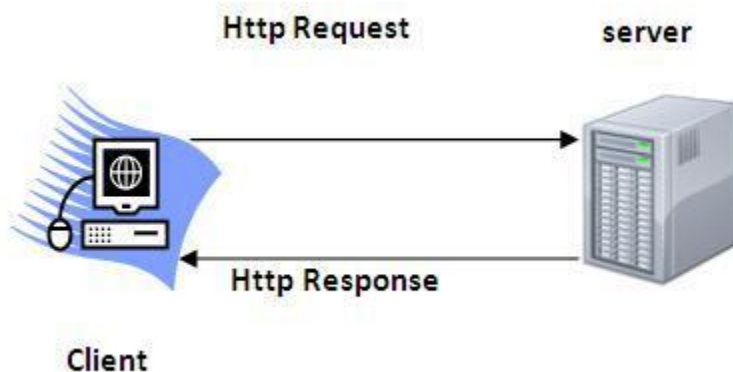
Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

There are many content types:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- video/quicktime etc.

HTTP (Hyper Text Transfer Protocol)

1. Http is the protocol that allows web servers and browsers to exchange data over the web.
2. It is a request response protocol.
3. Http uses reliable TCP connections by default on TCP port 80.
4. It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.



Http Request Methods

Every request has a header that tells the status of the client. There are many request methods. Get and Post requests are mostly used.

The http request methods are:

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

What is the difference between Get and Post?

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2) Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
3) Get request can be bookmarked	Post request cannot be bookmarked
4) Get request is idempotent . It means second request will be ignored until response of first request is delivered.	Post request is non-idempotent
5) Get request is more efficient and used more than Post	Post request is less efficient and used less than get.

Anatomy of Get Request

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what informations are sent to the server.

HTTP request FORMAT:-

Initial request line ex:- ...GET /one.html HTTP/1.1

Header line 1

Header line 2

Header line n

blank line

request body (optional)

Format of Initial Request line:-

GET one.html HTTP/1.1

here GET : Request method

one.html : Request URI

HTTP/1.1 : protocol/version no. used by the browser(client)

Format of the Header line:-

user-Agent: Mozilla/4.5

here user-Agent : Header name

Mozilla/4.5: Header value

Format of HTTP RESPONSE:-

Initial Response Line

Header line 1

Header line n

blank line

Response body

(optional)

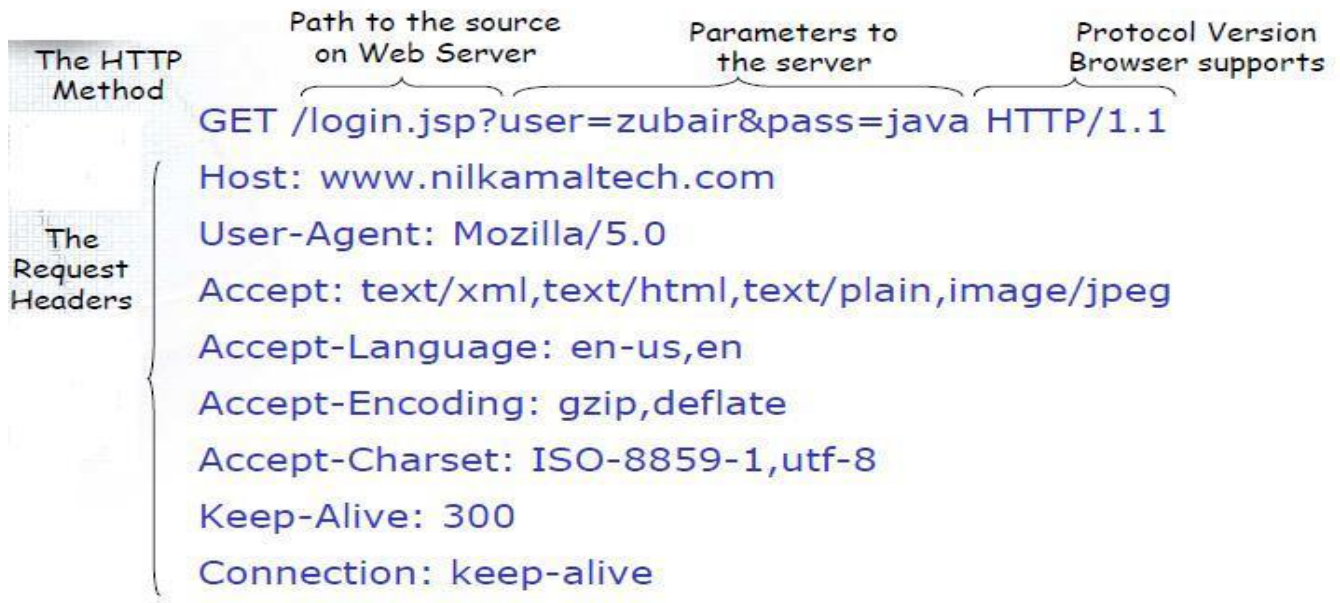
Format of Initial response line:-

HTTP/1.0 200 OK

Here HTTP/1.0 : protocol/version no used by the browser.

200 : status code

OK : status message



Anatomy of Post Request

As we know, in case of post request original data is sent in message body. Let's see how informations are passed to the server in case of post request.



Servlet API

The `javax.servlet` and `javax.servlet.http` packages represent interfaces and classes for servlet api.

The **`javax.servlet`** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **`javax.servlet.http`** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of `javax.servlet` package.

Interfaces in `javax.servlet` package

There are many interfaces in `javax.servlet` package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent

6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

UNIT-V

Lecture-4

Servlet LifeCycle:

A servlet follows a certain life cycle. The servlet life cycle is managed by the servlet container. The life cycle contains the following steps:

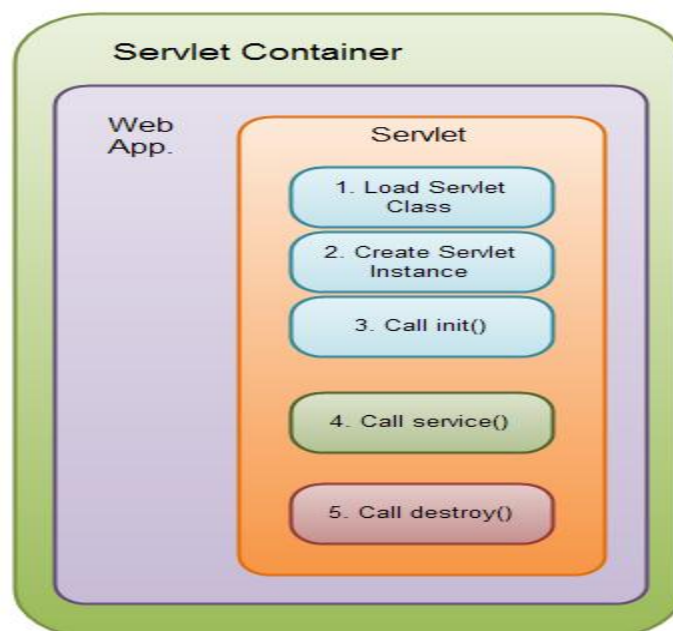
1. Load Servlet Class.
2. Create Instance of Servlet.
3. Call the servlets `init()` method.
4. Call the servlets `service()` method.
5. Call the servlets `destroy()` method.

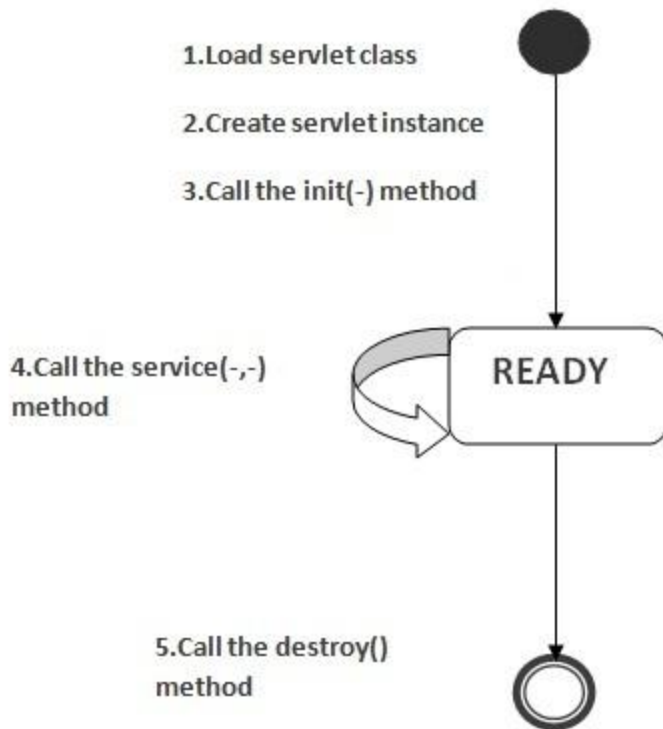
Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though.

See **web.xml Servlet Configuration** for more details about that.

Step 4 is executed multiple times - once for every HTTP request to the servlet. Step 5 is executed when the servlet container unloads the servlet. Each step is described in more detail below:

The Java Servlet life cycle





Load Servlet Class

Before a servlet can be invoked the servlet container must first load its class definition. This is done just like any other class is loaded.

Create Instance of Servlet

When the servlet class is loaded, the servlet container creates an instance of the servlet.

Typically, only a single instance of the servlet is created, and concurrent requests to the servlet are executed on the same servlet instance. This is really up to the servlet container to decide, though. But typically, there is just one instance.

Call the Servlets init() Method

When a servlet instance is created, its `init()` method is invoked. The `init()` method allows a servlet to initialize itself before the first request is processed.

You can specify init parameters to the servlet in the `web.xml` file. See [web.xml Servlet Configuration](#) for more details.

Call the Servlets service() Method

For every request received to the servlet, the `service()` method is called. For `HttpServlet` subclasses, one of the `doGet()`, `doPost()` etc. methods are typically called.

As long as the servlet is active in the servlet container, the `service()` method can be called. Thus, this step in the life cycle can be executed multiple times.

Call the Servlets destroy() Method

When a servlet is unloaded by the servlet container, its `destroy()` method is called. This step is only executed once, since a servlet is only unloaded once.

A servlet is unloaded by the container if the container shuts down, or if the container reloads the whole web application at runtime.

Steps to create a servlet example

1. [Steps to create the servlet using Tomcat server](#)
1. [Create a directory structure](#)
2. [Create a Servlet](#)
3. [Compile the Servlet](#)
4. [Create a deployment descriptor](#)
5. [Start the server and deploy the application](#)

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

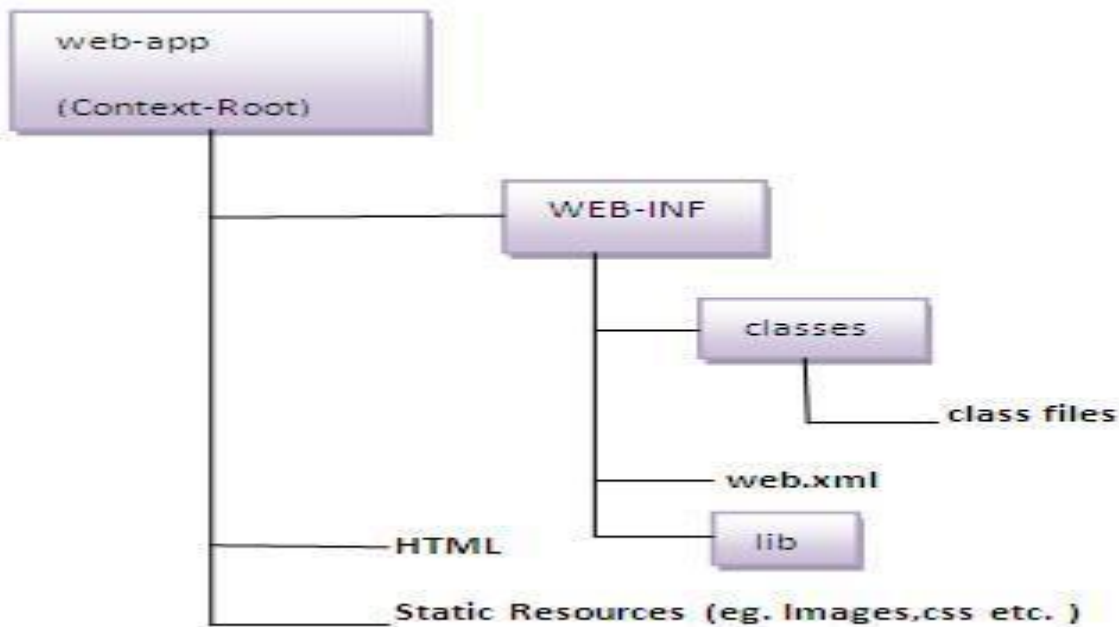
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.


```
C:\Windows\system32\cmd.exe
C:\Users\YELLASWAMY>cd E:\IIICSEA\cmrwebapp1
C:\Users\YELLASWAMY>e:
E:\IIICSEA\cmrwebapp1>tree/f
Folder PATH listing
Volume serial number is 3643-98F2
E:.
├── CMRCETWebApp1
│   ├── Home.html
│   └── WEB-INF
│       ├── web.xml
│       ├── classes
│       │   └── servlethtml.java
│       ├── com
│       │   └── cmrcet
│       │       ├── servletexample
│       │       └── servlethtml.class
│       └── lib
└── lib
E:\IIICSEA\cmrwebapp1>
```



2) Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface

2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

Servlethtml.java

```
package com.cmrcet.servletexample;

import javax.servlet.http.*;

import java.io.*;

import javax.servlet.*;

public class servlethtml extends HttpServlet

{

public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException

{

    res.setContentType("text/html");

    System.out.println("this is displayed in Tomcat Server");

    PrintWriter out=res.getWriter();

    out.println("<html><body>");
```

```

out.println("<center><b>");

out.println("<i>Welcome to CMR College of Egeineering & Technology</i>");

out.println("<b>WT LAB</b>");

out.println("</b></center>");

out.println("</body></html>");

}

}

```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

```
E:\IICSEA\cmrwebapp1\WEB-INF\classes>set classpath=E:\servlet-api.jar;.;
```

```
E:\IICSEA\cmrwebapp1\WEB-INF\classes>javac -d . *.java
```

```
E:\IICSEA\cmrwebapp1\WEB-INF\classes>
```

```
E:\IICSEA\cmrwebapp1\WEB-INF\classes>set classpath=E:\servlet-api.jar;.;
```

```
E:\IICSEA\cmrwebapp1\WEB-INF\classes>javac -d . *.java
```

```
E:\IICSEA\cmrwebapp1>tree/f
```

Folder PATH listing

Volume serial number is 3643-98F2

E:.

| Home.html

|

└── WEB-INF

| web.xml

|

└── classes

| | servlehtml.java

| |

| └── com

| └── cmrcet

| └── servletexample

| └── servlehtml.class

|

└── lib

4) Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

Web.xml

```
<web-app>
<servlet>
<servlet-name>servlethtml</servlet-name>
<servlet-class>com.cmrctet.servletexample.servlethtml</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>servlethtml</servlet-name>
<url-pattern>/getHomeHtml</url-pattern>
</servlet-mapping>
</web-app>
```

5) Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

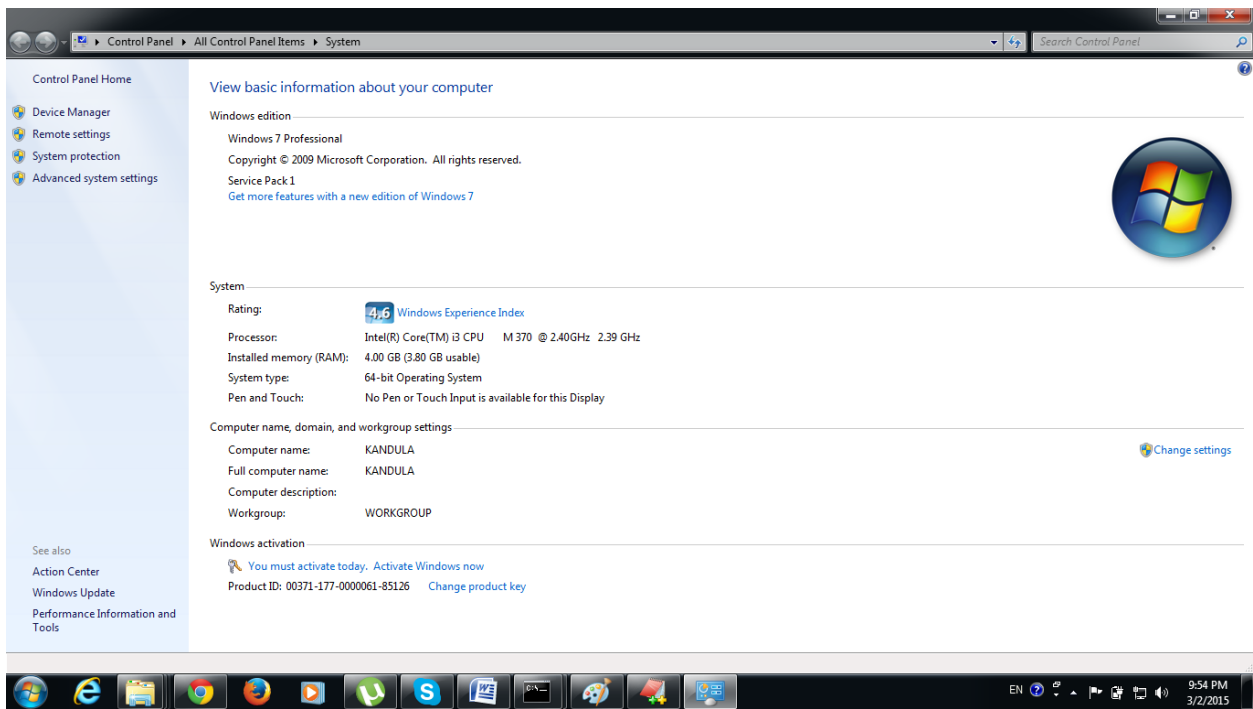
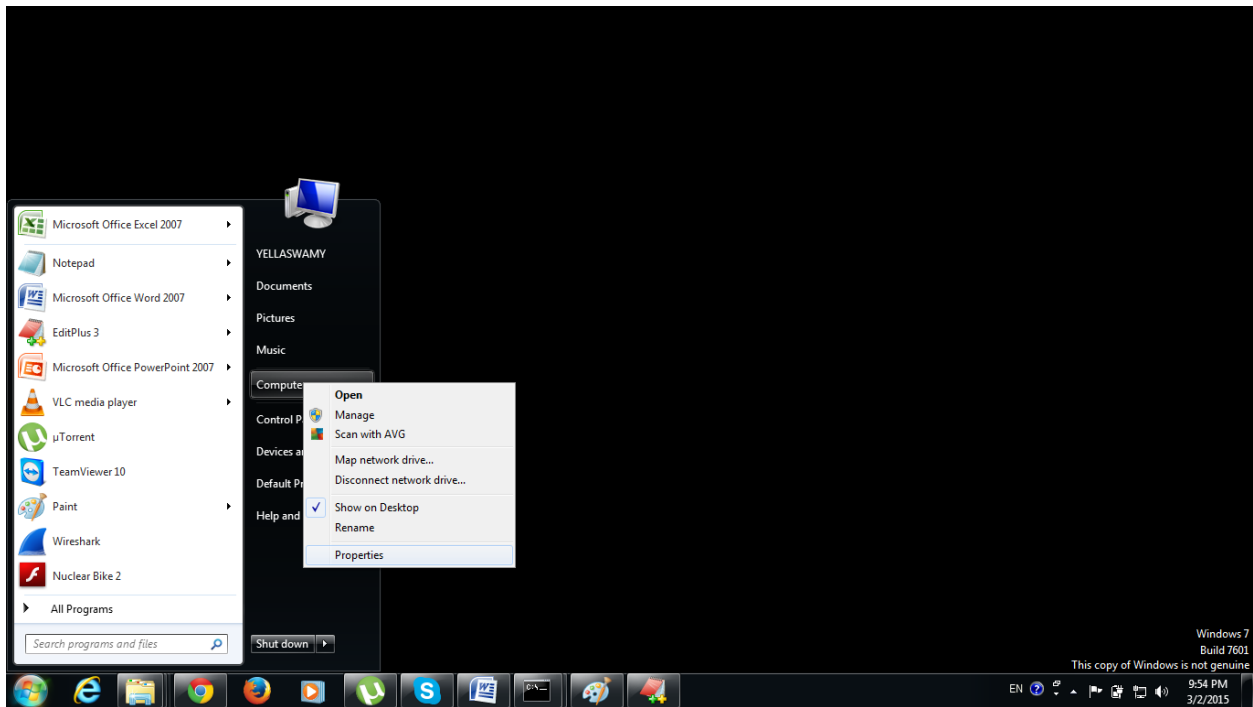
1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

1) How to set JAVA_HOME in environment variable?

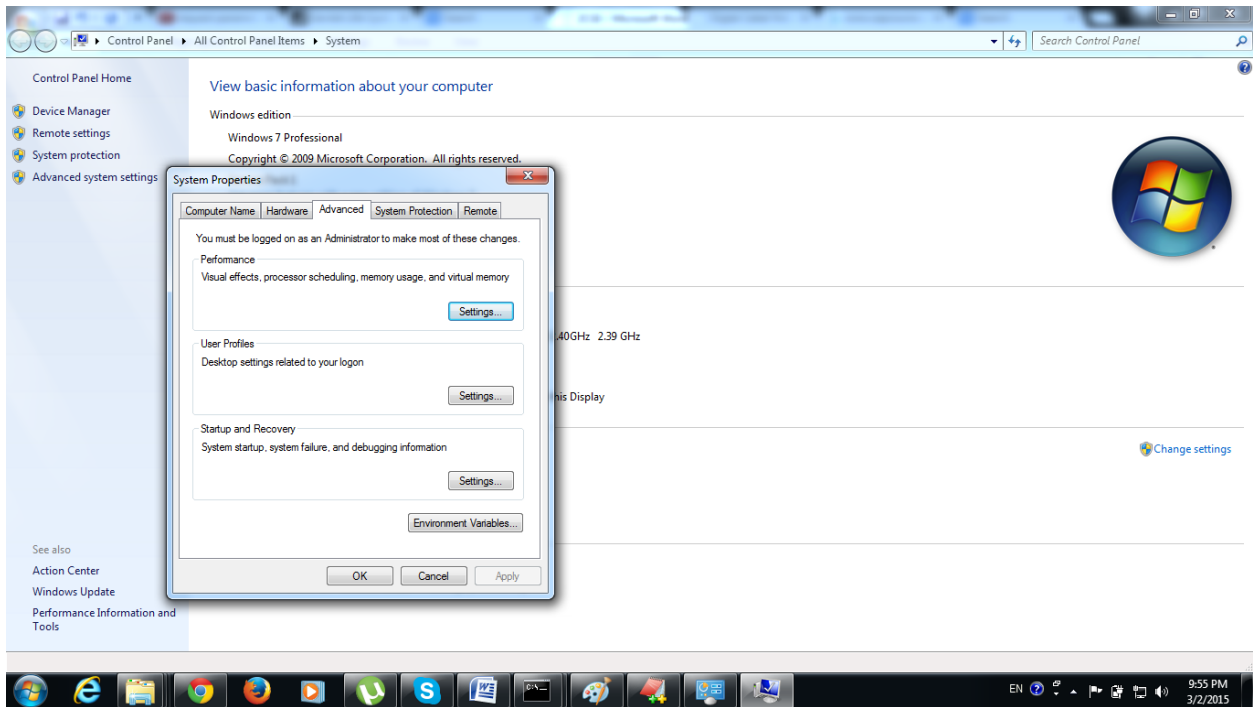
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

Go to My Computer properties -> Click on advanced tab then environment variables
-> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

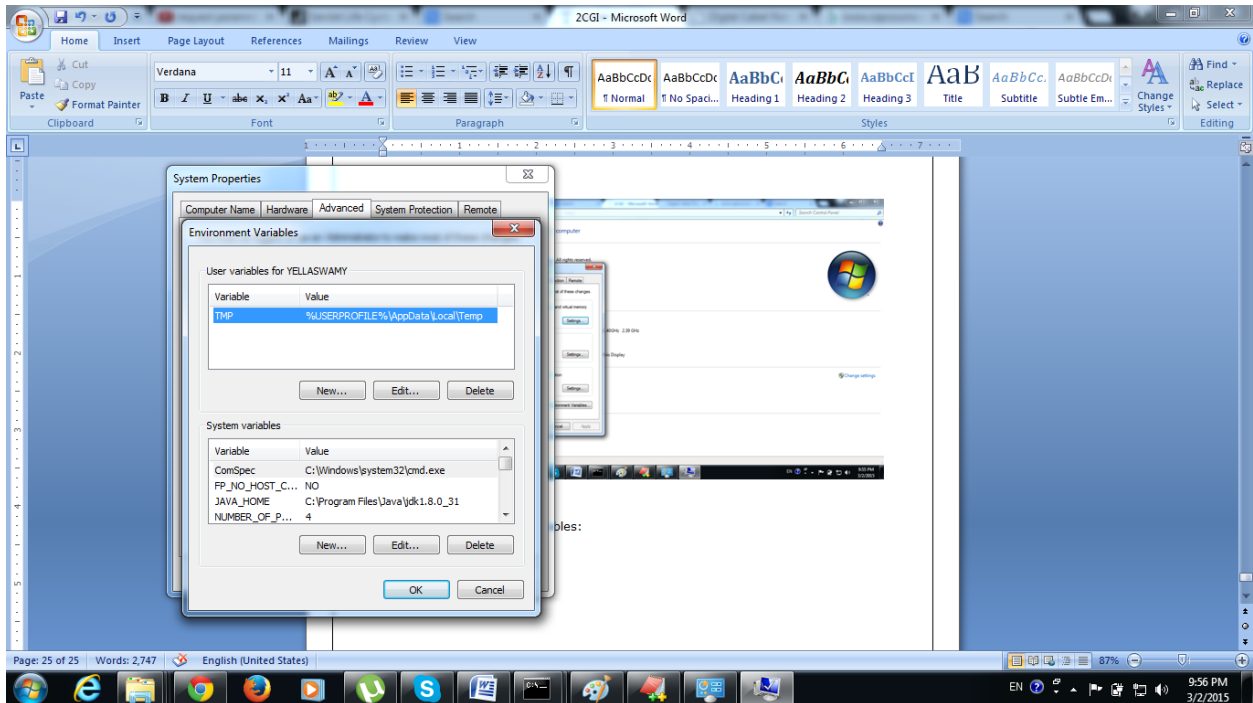
Go to My Computer properties:

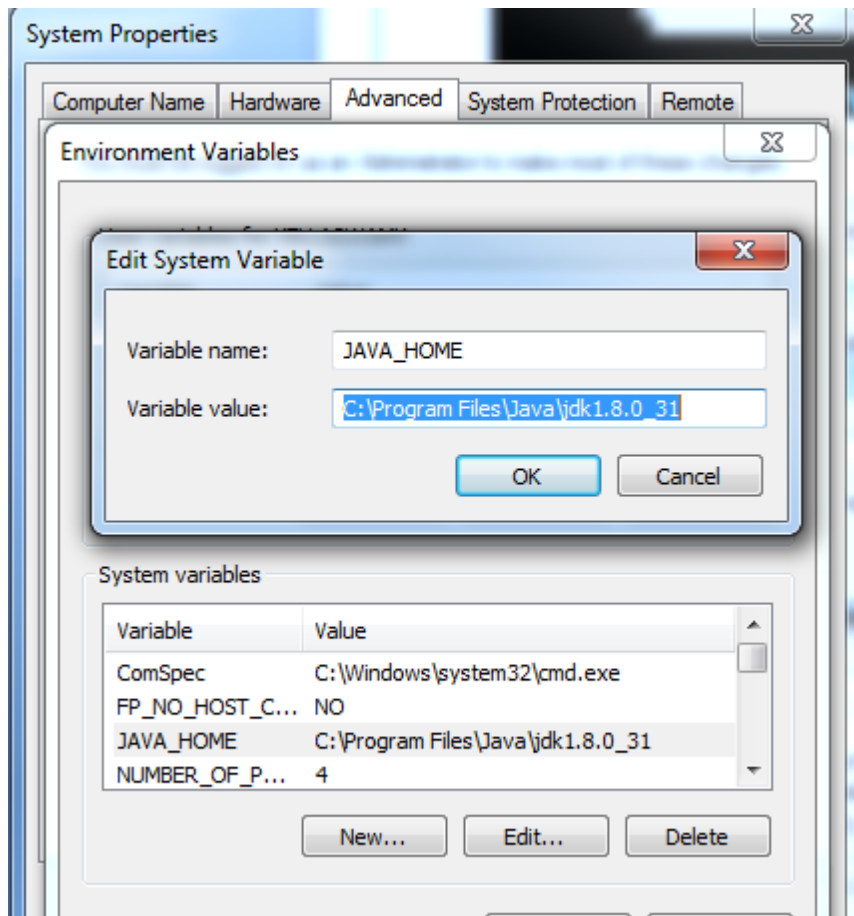


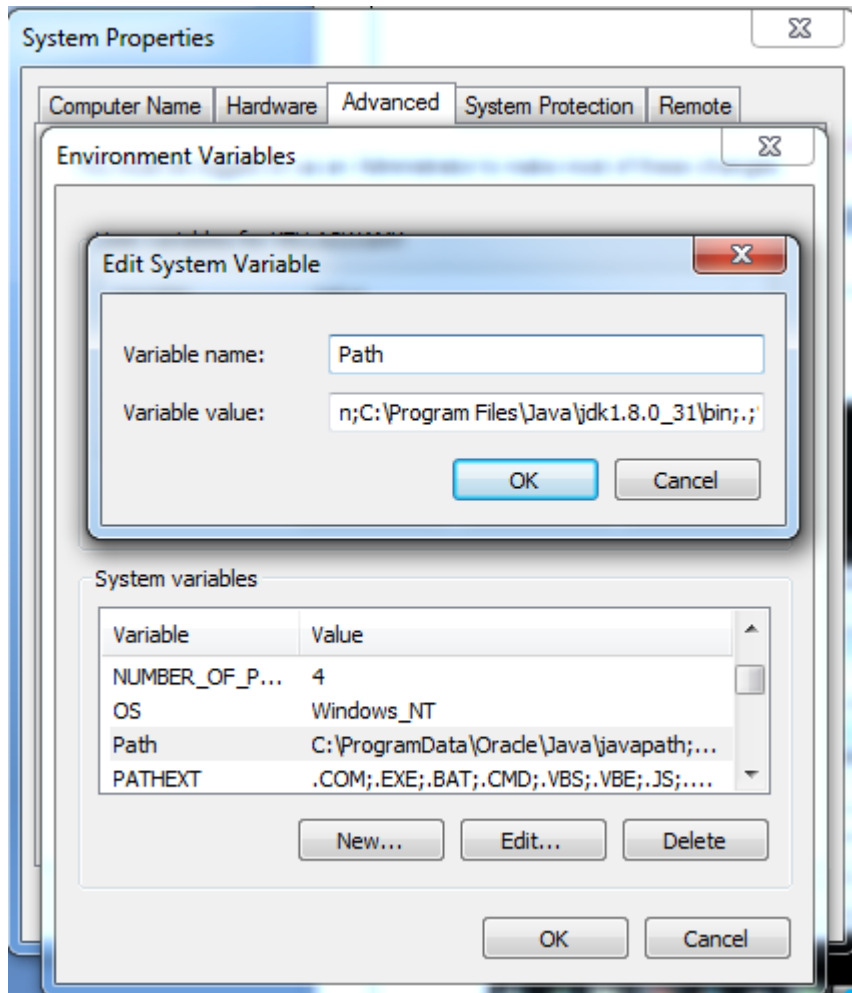
Click on advanced system settings tab then environment variables:



Click on Environmental Variables:





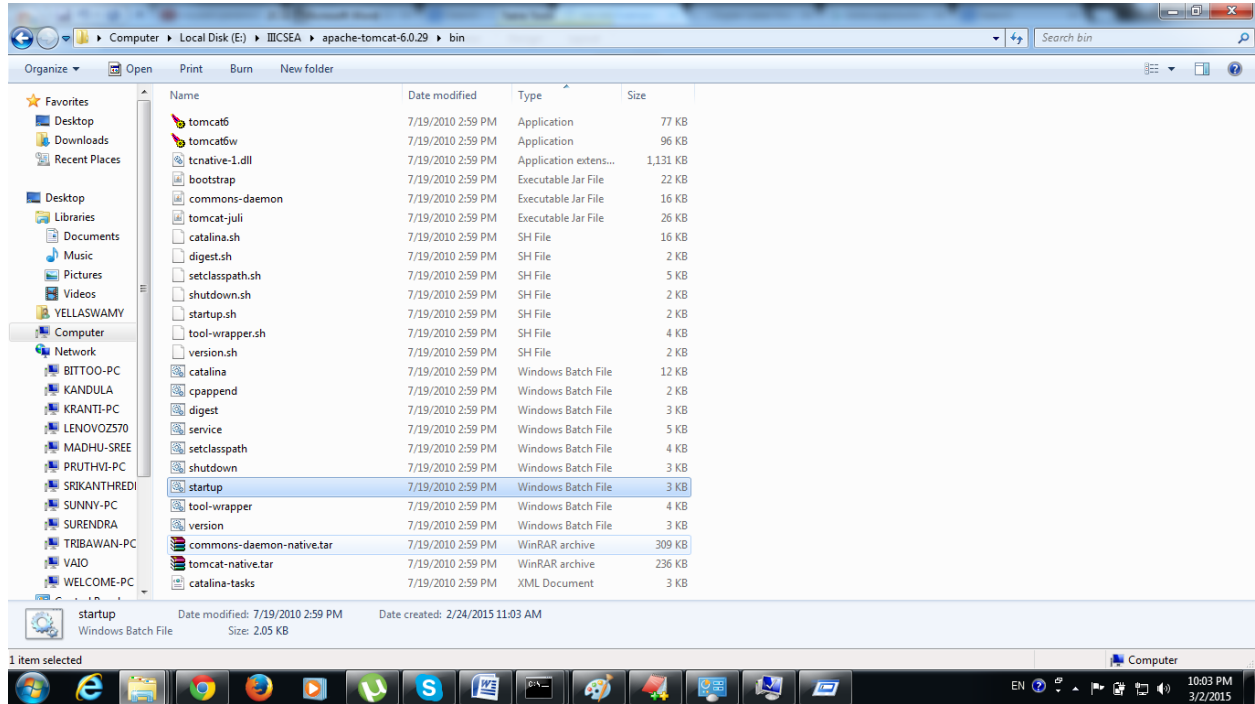


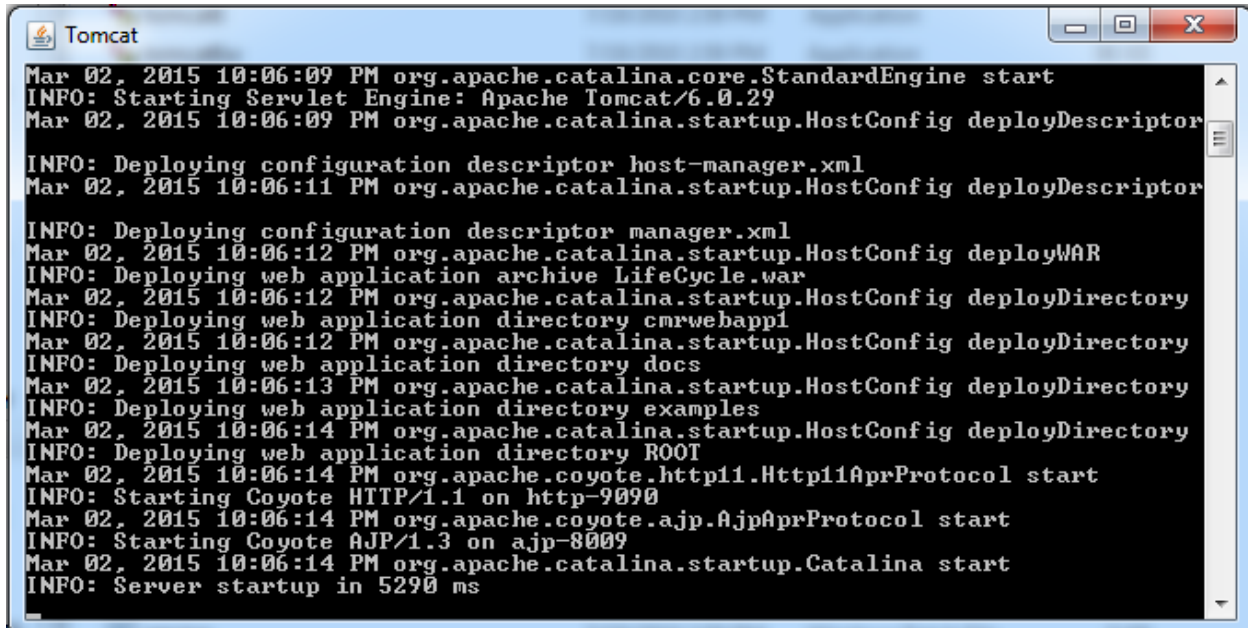
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

Note: There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

It is the example of apache tomcat that needs to extract only.





```
Tomcat
Mar 02, 2015 10:06:09 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.29
Mar 02, 2015 10:06:09 PM org.apache.catalina.startup.HostConfig deployDescriptor
INFO: Deploying configuration descriptor host-manager.xml
Mar 02, 2015 10:06:11 PM org.apache.catalina.startup.HostConfig deployDescriptor
INFO: Deploying configuration descriptor manager.xml
Mar 02, 2015 10:06:12 PM org.apache.catalina.startup.HostConfig deployWAR
INFO: Deploying web application archive LifeCycle.war
Mar 02, 2015 10:06:12 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory cmrwebapp1
Mar 02, 2015 10:06:12 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory docs
Mar 02, 2015 10:06:13 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory examples
Mar 02, 2015 10:06:14 PM org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory ROOT
Mar 02, 2015 10:06:14 PM org.apache.coyote.http11.Http11AprProtocol start
INFO: Starting Coyote HTTP/1.1 on http-9090
Mar 02, 2015 10:06:14 PM org.apache.coyote.ajp.AjpAprProtocol start
INFO: Starting Coyote AJP/1.3 on ajp-8009
Mar 02, 2015 10:06:14 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 5290 ms
```

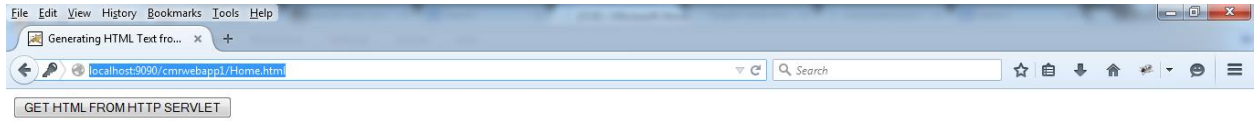
But there are several ways to deploy the project. They are as follows:

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

6) How to access the servlet

Open Browser and specify the following URL:

<http://localhost:9090/cmrwebapp1/Home.html>



LifeCycleServlet Example:

```
package com.cmrcet.lifecycleex;
import javax.servlet.*;
import java.io.*;
public class LifeCycle implements Servlet
{
private ServletConfig myconfig;
public void init(ServletConfig sc)
{
this.myconfig=sc;
System.out.println("In Init() Method");

}

public void service(ServletRequest myreq,ServletResponse myres)throws
ServletException,IOException
{

System.out.println("In service() Method");
PrintWriter out=myres.getWriter();
out.println("Hi,I am from Tomcat Server");
}

public void destroy()
{
System.out.println("In destroy() Method");
}

public String getServletInfo()
{
return "Developed by CMRCET";
}
public ServletConfig getServletConfig()
{
return myconfig;
}

}
```

Web.xml

```
<web-app>
<servlet>
<servlet-name>LifeCycle</servlet-name>
<servlet-class>com.cmrctet.lifecycleex.LifeCycle</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>LifeCycle</servlet-name>
<url-pattern>/lifecycle</url-pattern>
</servlet-mapping>
</web-app>
```

Lifecycle.html

```
<!doctype html>
<html lang="en">
<head>

<title>Life Cycle Servlet</title>
</head>
<body>
<form action="lifecycle">
<input type="submit" value="Invoke LifeCycle Servlet"/>
</form>

</body>
</html>
```

Note:Arrrane in Directory structure and compile and deploy in Tomcat server and Test

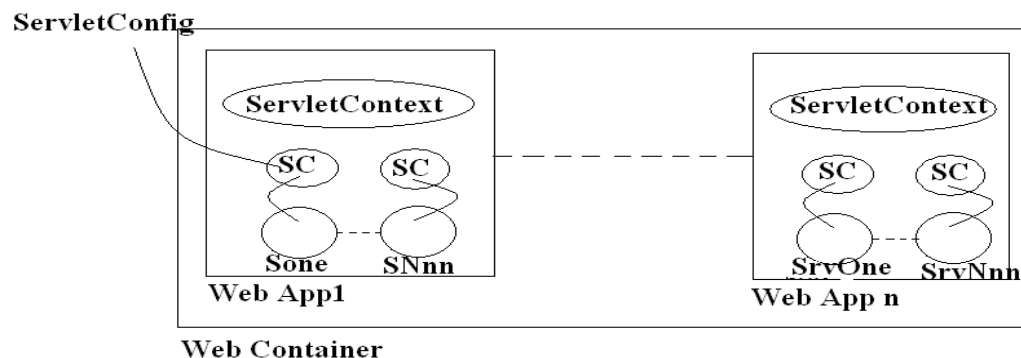
```
C:\Windows\system32\cmd.exe
E:\IICSEA\LifeCycleEx>tree/f
Folder PATH listing
Volume serial number is 3643-98F2
E:
| Lifecycle.html
| LifeCycle.war
| WEB-INF
| | web.xml
| | classes
| | | LifeCycle.java
| | | com
| | | | cmrcet
| | | | | lifecycleex
| | | | | LifeCycle.class
E:\IICSEA\LifeCycleEx>
```


UNIT-V

Lecture-5

ServletContext and ServletConfig:

ServletContext and ServletConfig are the interfaces provided by java soft as part of servlet-api. It is the responsibility of the web container vendors to provide the classes implementing the ServletConfig interface and the ServletContext interface.



1. We can run multiple web applications as part of a web container.
2. While starting a web application, Web container is responsible for creating a ServletContext object.
3. Web Container is responsible for removing/destroying the ServletContext object while stopping the web application.
4. There will be only one ServletContext object for every web application. The ServletContext object is also known as an application object.
5. There can be multiple servlets in a web application.
6. Web Container is responsible for maintaining a ServletConfig object for every servlet (there can be multiple ServletConfig objects in a web application).
7. ServletConfig can be used to get the information about the Initialization parameters.
8. ServletContext object can be used to get the information about the Context Parameters.
9. The Context parameters can be accessed by every Servlet that is the part of web application and the initialization parameters can be accessed only by the servlet for which the parameter is provided.

ServletContext Interface

1. [ServletContext Interface](#)
2. [Usage of ServletContext Interface](#)
3. [Methods of ServletContext interface](#)
4. [How to get the object of ServletContext](#)
5. [Syntax to provide the initialization parameter in Context scope](#)
6. [Example of ServletContext to get initialization parameter](#)
7. [Example of ServletContext to get all the initialization parameter](#)

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

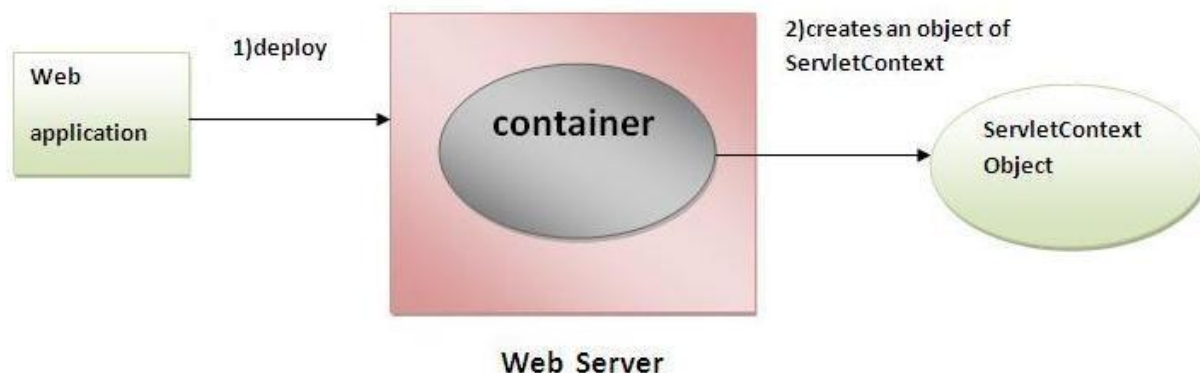
Advantage of ServletContext

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

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name, Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

How to get the object of ServletContext interface

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.
2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

Syntax of getServletContext() method

1. **public** ServletContext getServletContext()

Example of getServletContext() method

```
//We can get the ServletContext object from ServletConfig object
ServletContext application=getServletConfig().getServletContext();
```

```
//Another convenient way to get the ServletContext object
ServletContext application=getServletContext();
```

Syntax to provide the initialization parameter in Context scope

The **context-param** element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and param-value defines its value.

```
<web-app>
.....

<context-param>
  <param-name>parametername</param-name>
  <param-value>parametervalue</param-value>
</context-param>
.....
</web-app>
```

Example of ServletContext to get the initialization parameter

In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element.

Let's see the simple example:

SCEExample.java

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

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

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

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

pw.close();

}
}
```

```
E:\M\CSEA\UNIT5_Servlets\3ServletContextExample\WEB-INF\classes\SCEExample.java - EditPlus
File Edit View Search Document Project Tools Browser ZC Window Help
1 import java.io.*;
2 import javax.servlet.*;
3 import javax.servlet.http.*;
4
5
6 public class SCEExample extends HttpServlet
7 {
8     public void doGet(HttpServletRequest req,HttpServletResponse res)
9     throws ServletException,IOException
10    {
11        res.setContentType("text/html");
12        PrintWriter pw=res.getWriter();
13
14        //creating ServletContext object
15        ServletContext mycontext=getServletContext();
16
17        //Getting the value of the initialization parameter and printing it
18        String driverName=mycontext.getInitParameter("DatabaseDriver");
19        pw.println("Driver name is="+driverName);
20
21        pw.close();
22    }
23 }
24 }
```

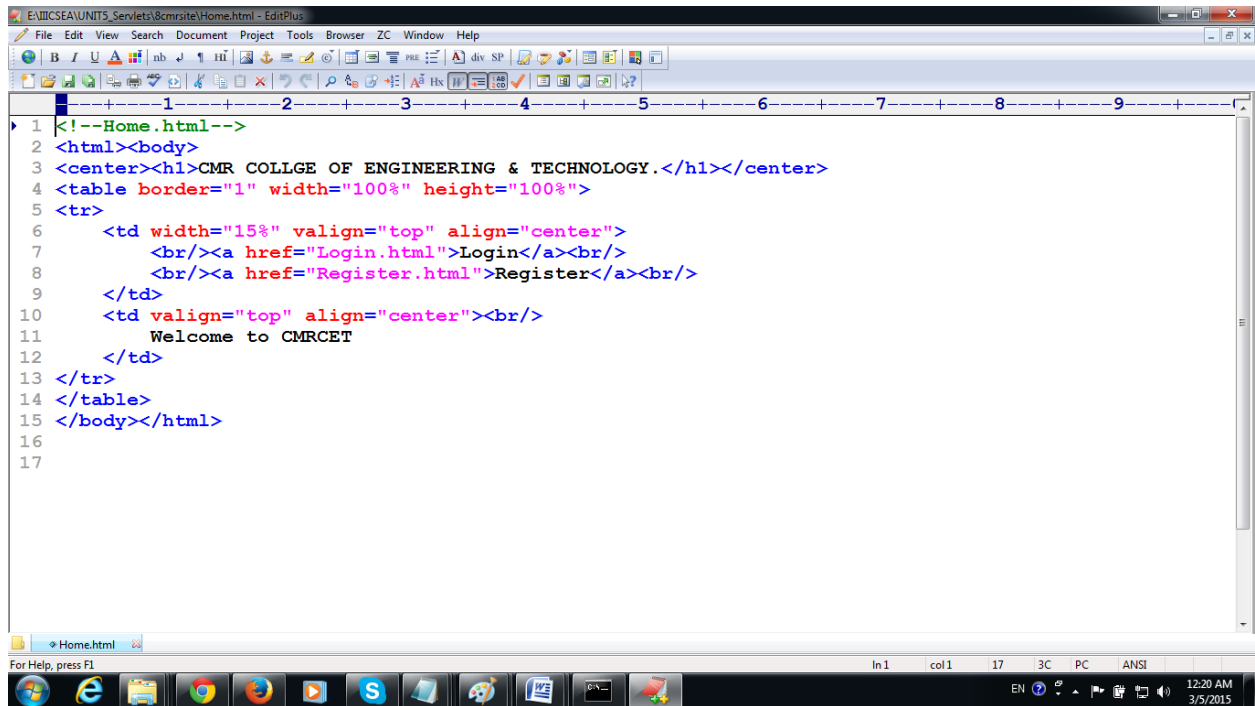
For Help, press F1 In 18 col 61 24 22 PC ANSI 11:44 3/4/2015

Web.xml

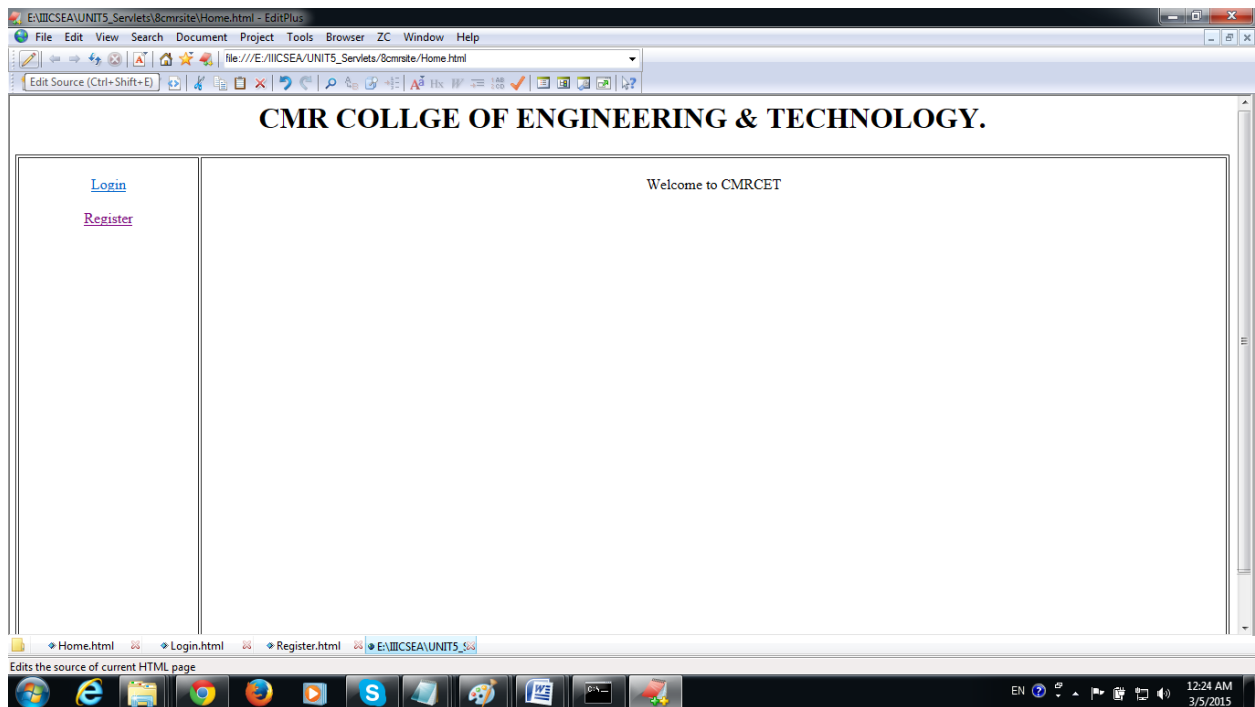
```
E:\M\CSEA\UNIT5_Servlets\3ServletContextExample\WEB-INF\web.xml - EditPlus
File Edit View Search Document Project Tools Browser ZC Window Help
1 <web-app>
2     <context-param>
3         <param-name>DatabaseDriver</param-name>
4         <param-value>oracle.jdbc.driver.OracleDriver</param-value>
5     </context-param>
6     <context-param>
7         <param-name>username</param-name>
8         <param-value>yellaswamy</param-value>
9     </context-param>
10    <context-param>
11        <param-name>password</param-name>
12        <param-value>yellaswamy</param-value>
13    </context-param>
14 </context-param>
15    <servlet>
16        <servlet-name>SCEExample</servlet-name>
17        <servlet-class>SCEExample</servlet-class>
18    </servlet>
19    <servlet-mapping>
20        <servlet-name>SCEExample</servlet-name>
21        <url-pattern>/mycontext</url-pattern>
22    </servlet-mapping>
23 </web-app>
24
```

For Help, press F1 In 4 col 49 24 3C PC ANSI 11:42 PM 3/4/2015

Write Servlet program for Login and Registration to Connect to Database using ServletContext:



```
1 <!--Home.html-->
2 <html><body>
3 <center><h1>CMR COLLEGE OF ENGINEERING & TECHNOLOGY.</h1></center>
4 <table border="1" width="100%" height="100%">
5 <tr>
6 <td width="15%" valign="top" align="center">
7 <br/><a href="Login.html">Login</a><br/>
8 <br/><a href="Register.html">Register</a><br/>
9 </td>
10 <td valign="top" align="center"><br/>
11 Welcome to CMRCET
12 </td>
13 </tr>
14 </table>
15 </body></html>
16
17
```

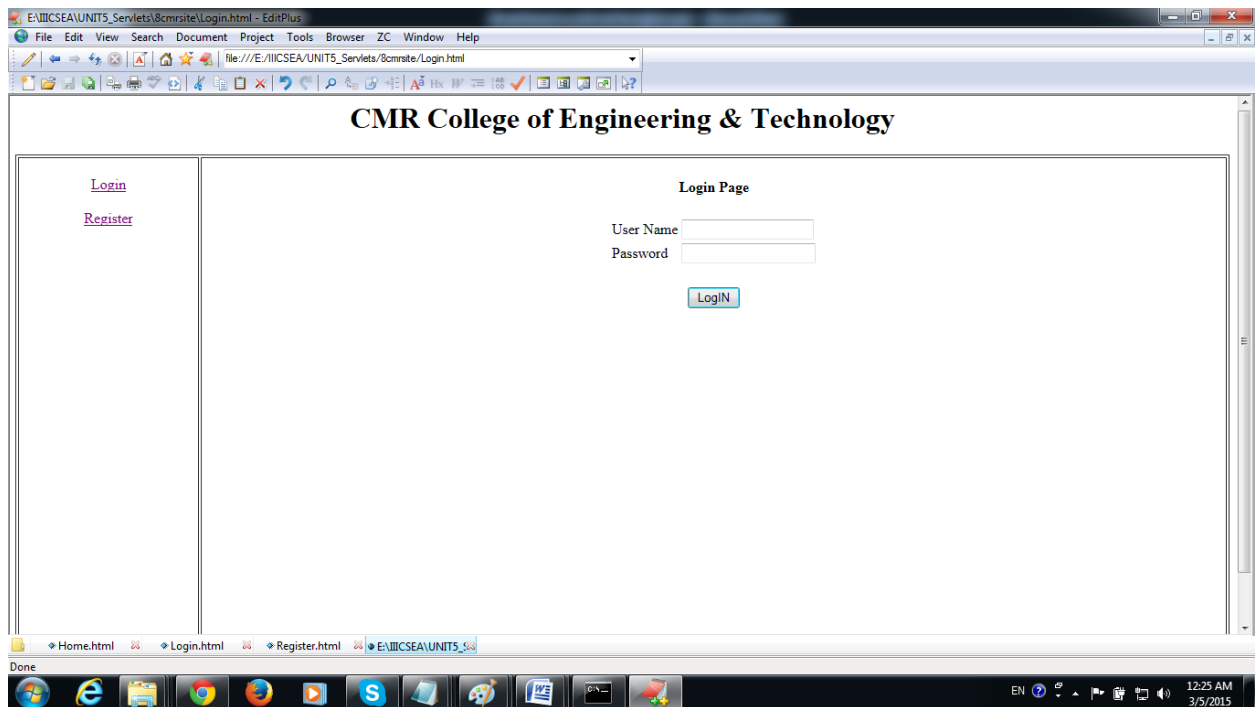


Login.html

```

<!--Login.html-->
<html> <body>
<center><h1>CMR College of Engineering & Technology</h1></center>
<table border="1" width="100%" height="100%">
<tr>
  <td width="15%" valign="top" align="center">
    <br/><a href="Login.html">Login</a><br/>
    <br/><a href="Register.html">Register</a><br/>
  </td>
  <td valign="top" align="center"><br/>
    <form action="login"><table>
      <tr>
        <td colspan="2" align="center"><b>Login Page</b></td>
      </tr>
      <tr>
        <td colspan="2" align="center"><b>&nbsp;</b></td>
      </tr>
      <tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
      </tr>
      <tr>
        <td>Password</td>
        <td><input type="password" name="pass"/></td>
      </tr>
      <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
      </tr>
      <tr>
        <td colspan="2" align="center"><input type="submit" value="LogIN"/></td>
      </tr>
    </table></form>
  </td>
</tr>
</table>
</body></html>

```

Register.html

`<!--Register.html-->`

`<html> <body>`

`<center><h1>CMR COLLEGE OF ENGINEERING & TECHNOLOGY</h1></center>`

`<table border="1" width="100%" height="100%">`

`<tr>`

`<td width="15%" valign="top" align="center">`

`
Login
`

`
Register
`

`</td>`

`<td valign="top" align="center">
`

`<form action="register"><table>`

`<tr>`

`<td colspan="2" align="center">Registration Page</td>`

`</tr>`

`<tr>`

`<td colspan="2" align="center"> </td>`

`</tr>`

`<tr>`

`<td>User Name</td>`

`<td><input type="text" name="uname"/></td>`

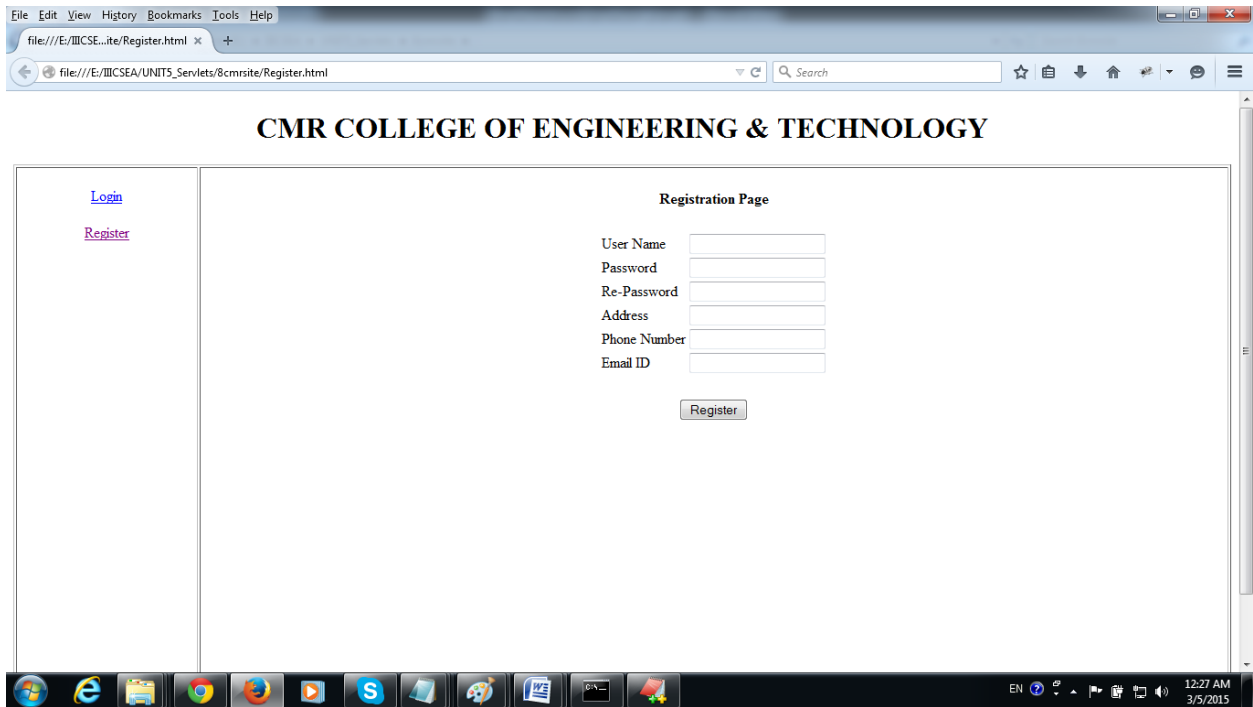
`</tr>`

`<tr>`

```

        <td>Password</td>
        <td><input type="password" name="pass"/></td>
    </tr>
    <tr>
        <td>Re-Password</td>
        <td><input type="password" name="repass"/></td>
    </tr>
    <tr>
        <td>Address</td>
        <td><input type="text" name="addr"/></td>
    </tr>
    <tr>
        <td>Phone Number</td>
        <td><input type="text" name="phno"/></td>
    </tr>
    <tr>
        <td>Email ID</td>
        <td><input type="text" name="email"/></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td>&nbsp;</td>
    </tr>
    <tr>
        <td colspan="2" align="center"><input type="submit" value="Register"/></td>
    </tr>
</table></form>
</td>
</tr>
</table>
</body></html>

```



Web.xml

```
<?xml version="1.0"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app>
  <context-param>
    <param-name>driverClassName</param-name>
    <param-value>oracle.jdbc.driver.OracleDriver</param-value>
  </context-param>
  <context-param>
    <param-name>url</param-name>
    <param-value>
      jdbc:oracle:thin:@localhost:1521:xe
    </param-value>
  </context-param>
  <servlet>
    <servlet-name>ls</servlet-name>
    <servlet-class>com.yellaswamy.servlets.LoginServlet</servlet-class>
    <init-param>
      <param-name>dbuser</param-name>
      <param-value>yellaswamy</param-value>
    </init-param>
  </servlet>
</web-app>
```

```

        </init-param>
        <init-param>
            <param-name>dbpass</param-name>
            <param-value>kandula</param-value>
        </init-param>
        <init-param>
            <param-name>sqlstatement</param-name>
            <param-value>
                select * from userdetails where uname=? and pass=?
            </param-value>
        </init-param>
    </servlet>

<servlet>
    <servlet-name>rs</servlet-name>
    <servlet-class>
        com.yellaswamy.servlets.RegistrationServlet
    </servlet-class>

    <init-param>
        <param-name>dbuser</param-name>
        <param-value>yellaswamy</param-value>
    </init-param>
    <init-param>
        <param-name>dbpass</param-name>
        <param-value>yellaswamy</param-value>
    </init-param>
    <init-param>
        <param-name>sqlstatement</param-name>
        <param-value>
            insert into userdetails values(?,?,?,?)
        </param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>ls</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>rs</servlet-name>
    <url-pattern>/register</url-pattern>

```

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

```
<welcome-file-list>
```

```
<welcome-file>Home.html</welcome-file>
```

```
</welcome-file-list>
```

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

UNIT-V

Lecture-6

Session Tracking in Servlets

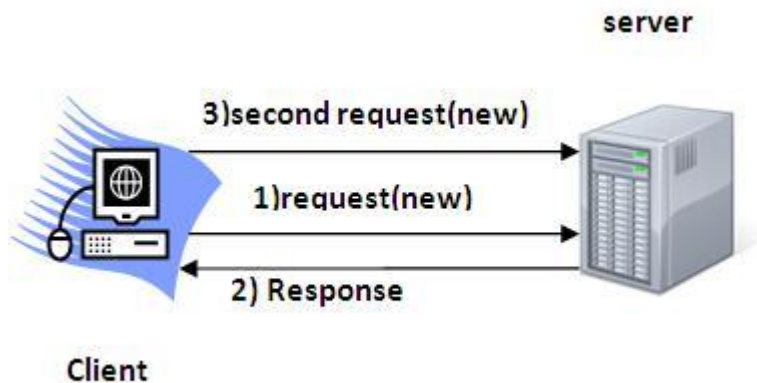
1. [Session Tracking](#)
2. [Session Tracking Techniques](#)

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

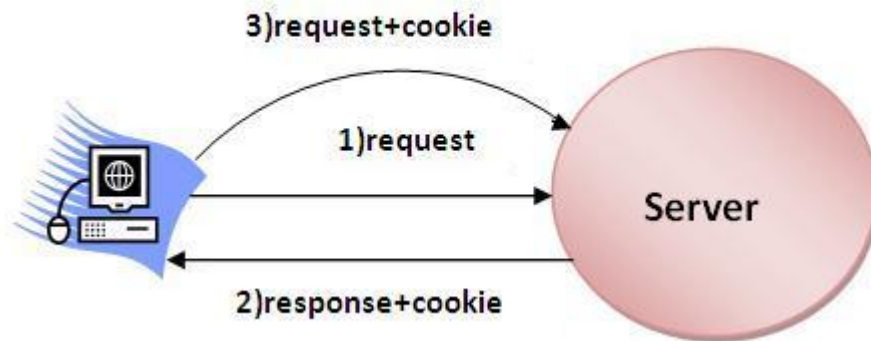
Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

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

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
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
```

How to get Cookies?

Let's see the simple code to get all the cookies.

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

Servlet Login and Logout Example using Cookies

A **cookie** is a kind of information that is stored at client side.

In the previous page, we learned a lot about cookie e.g. how to create cookie, how to delete cookie, how to get cookie etc.

Here, we are going to create a login and logout example using servlet cookies.

In this example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

index.html

```
<!DOCTYPE html>
<html>
<head>

<title>Servlet Login Example</title>
</head>
<body>

<h1>Welcome to Login Application by Cookie</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
<a href="ProfileServlet">Profile</a>
```

```
</body>
</html>
```

Link.html

```
<!doctype html>
<html lang="en">
<head>

  <title>Links</title>
</head>
<body>
  <a href="login.html">Login</a> |
  <a href="LogoutServlet">Logout</a> |
  <a href="ProfileServlet">Profile</a>
  <hr>
</body>
</html>
```

Login.html

```
<!doctype html>
<html lang="en">
<head>

  <title>Login</title>
</head>
<body>
  <form action="LoginServlet" method="post">
  Name:<input type="text" name="name"><br>
  Password:<input type="password" name="password"><br>
  <input type="submit" value="login">
  </form>
</body>
</html>
```

//LoginServlet.java

```
package yellaswamy;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

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

        if(password.equals("admin123")){
            out.print("You are successfully logged in!");
            out.print("<br>Welcome, "+name);

            Cookie ck=new Cookie("name",name);
            response.addCookie(ck);
        }else{
            out.print("sorry, username or password error!");
            request.getRequestDispatcher("login.html").include(request, response);
        }

        out.close();
    }
}
```

// LogoutServlet.java

```
package yellaswamy;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LogoutServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck=new Cookie("name","");
        ck.setMaxAge(0);
        response.addCookie(ck);

        out.print("you are successfully logged out!");
    }
}
```

```

// ProfileServlet.java
package yellaswamy;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

        request.getRequestDispatcher("link.html").include(request, response);

        Cookie ck[]=request.getCookies();
        if(ck!=null){
            String name=ck[0].getValue();
            if(!name.equals("") || name!=null){
                out.print("<b>Welcome to Profile</b>");
                out.print("<br>Welcome, "+name);
            }
        }else{
            out.print("Please login first");
            request.getRequestDispatcher("login.html").include(request, response);
        }
        out.close();
    }
}

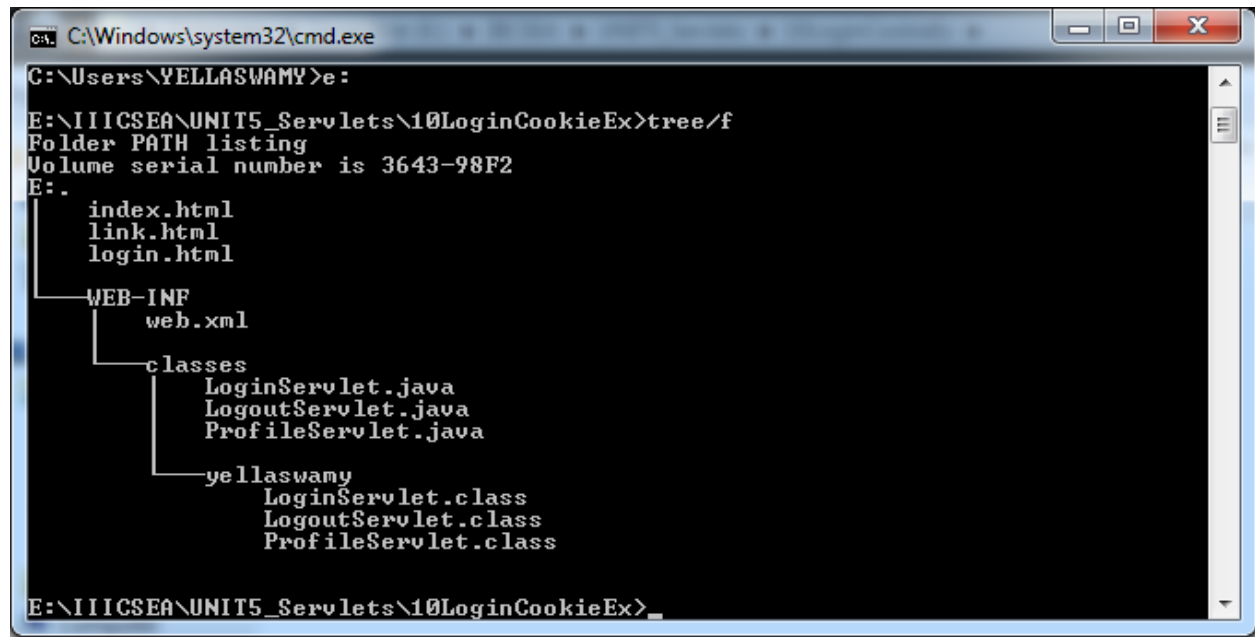
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>

  <servlet>
    <description></description>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>yellaswamy.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>ProfileServlet</display-name>
    <servlet-name>ProfileServlet</servlet-name>
    <servlet-class>yellaswamy.ProfileServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ProfileServlet</servlet-name>
    <url-pattern>/ProfileServlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <description></description>
    <display-name>LogoutServlet</display-name>
    <servlet-name>LogoutServlet</servlet-name>
    <servlet-class>yellaswamy.LogoutServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/LogoutServlet</url-pattern>
  </servlet-mapping>
</web-app>
```


Directory Structure:



```
C:\Windows\system32\cmd.exe
C:\Users\YELLASWAMY>e:
E:\IIICSEA\UNIT5_Servlets\10LoginCookieEx>tree/f
Folder PATH listing
Volume serial number is 3643-98F2
E:
├── index.html
├── link.html
├── login.html
├── WEB-INF
│   ├── web.xml
│   └── classes
│       ├── LoginServlet.java
│       ├── LogoutServlet.java
│       └── ProfileServlet.java
└── yellaswamy
    ├── LoginServlet.class
    ├── LogoutServlet.class
    └── ProfileServlet.class

E:\IIICSEA\UNIT5_Servlets\10LoginCookieEx>
```

Output:



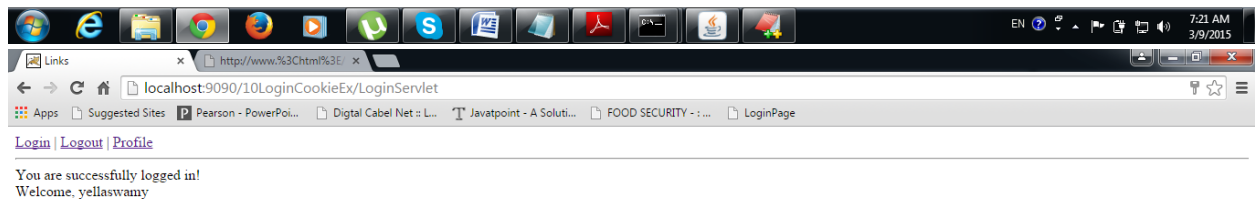
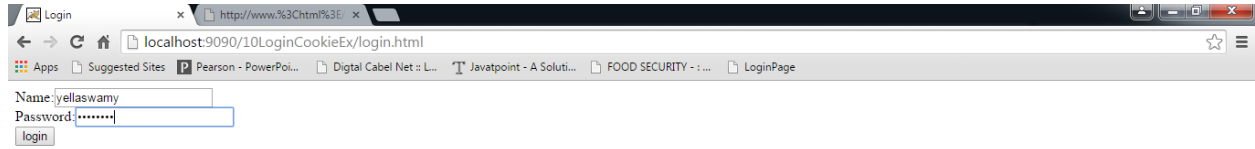
Welcome to Login Application by Cookie

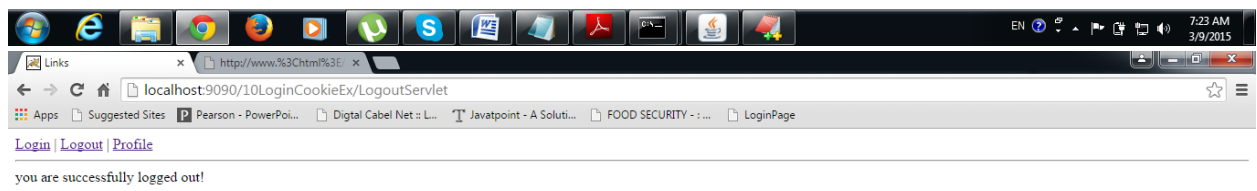
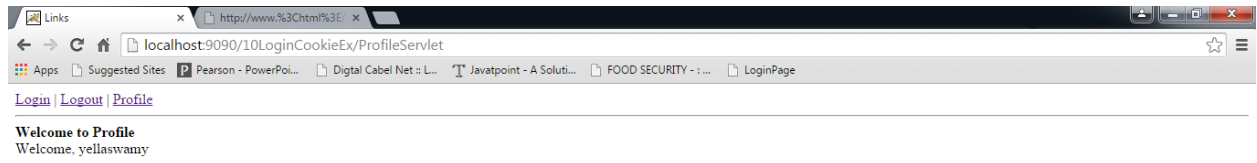
[Login](#) | [Logout](#) | [Profile](#)



Username=yellaswamy

Password=admin123





2) Hidden Form Field

1. [Hidden Form Field](#)
2. [Example of Hidden Form Field](#)

In case of Hidden Form Field a **hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

1. `<input type="hidden" name="uname" value="Vimal Jaiswal">`

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

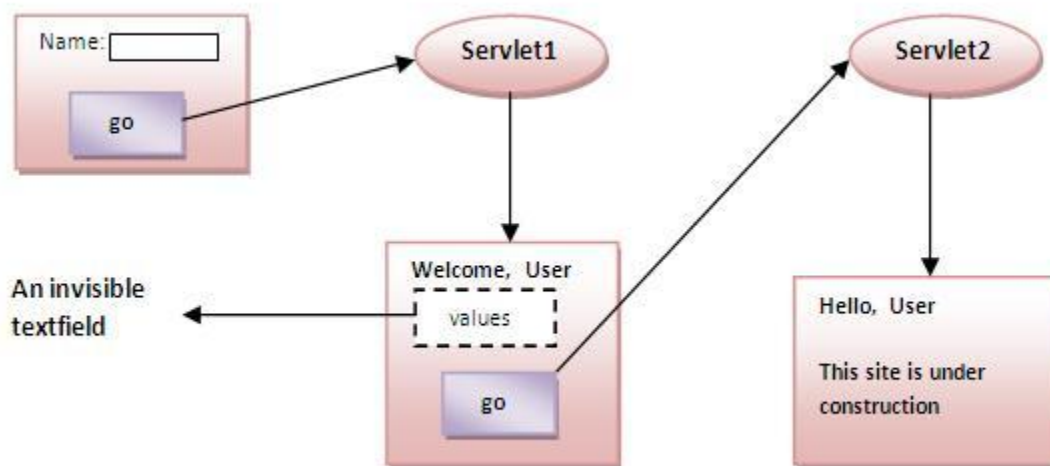
1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
 2. Extra form submission is required on each pages.
 3. Only textual information can be used.
-

Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden textfield and getting that value from another servlet.



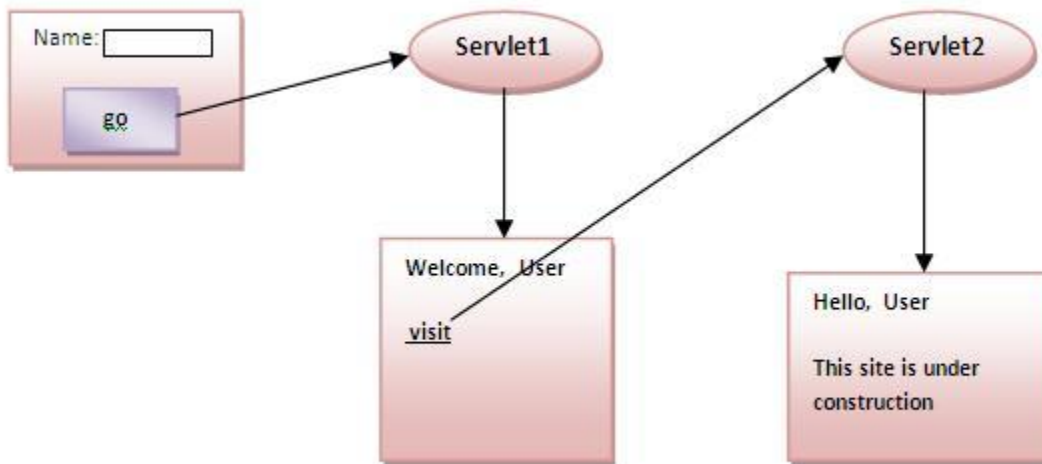
3)URL Rewriting

1. [URL Rewriting](#)
2. [Advantage of URL Rewriting](#)
3. [Disadvantage of URL Rewriting](#)
4. [Example of URL Rewriting](#)

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

```
url?name1=value1&name2=value2&??
```

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use `getParameter()` method to obtain a parameter value.



Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Example of using URL Rewriting

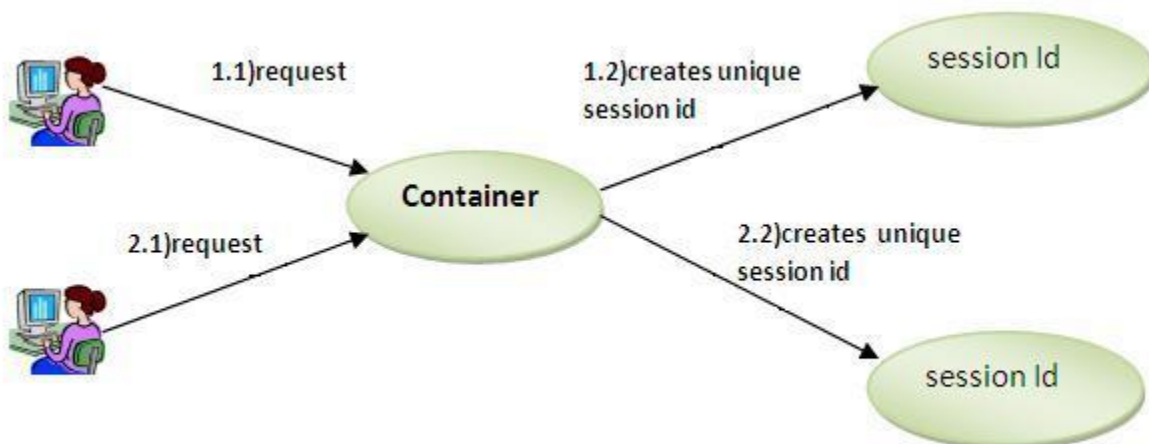
In this example, we are maintaining the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

4) HttpSession interface

1. [HttpSession interface](#)
2. [How to get the HttpSession object](#)
3. [Commonly used methods of HttpSession interface](#)
4. [Example of using HttpSession](#)

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):** Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the `setAttribute()` method of HttpSession interface and to get the attribute, we have used the `getAttribute` method.

Lecture-7

RequestDispatcher in Servlet

1. [RequestDispatcher Interface](#)
2. [Methods of RequestDispatcher interface](#)
 1. [forward method](#)
 2. [include method](#)
3. [How to get the object of RequestDispatcher](#)
4. [Example of RequestDispatcher interface](#)

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.

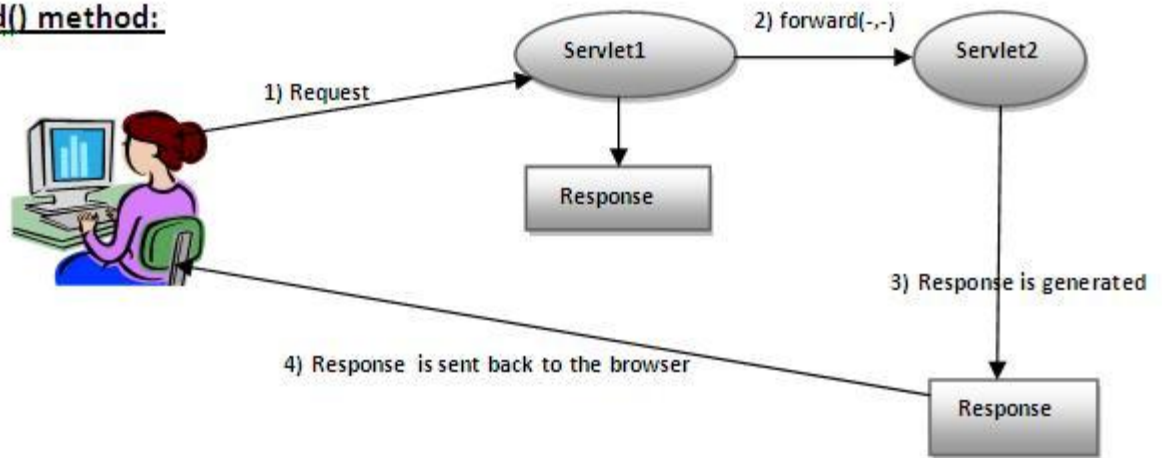
There are two methods defined in the RequestDispatcher interface.

Methods of RequestDispatcher interface

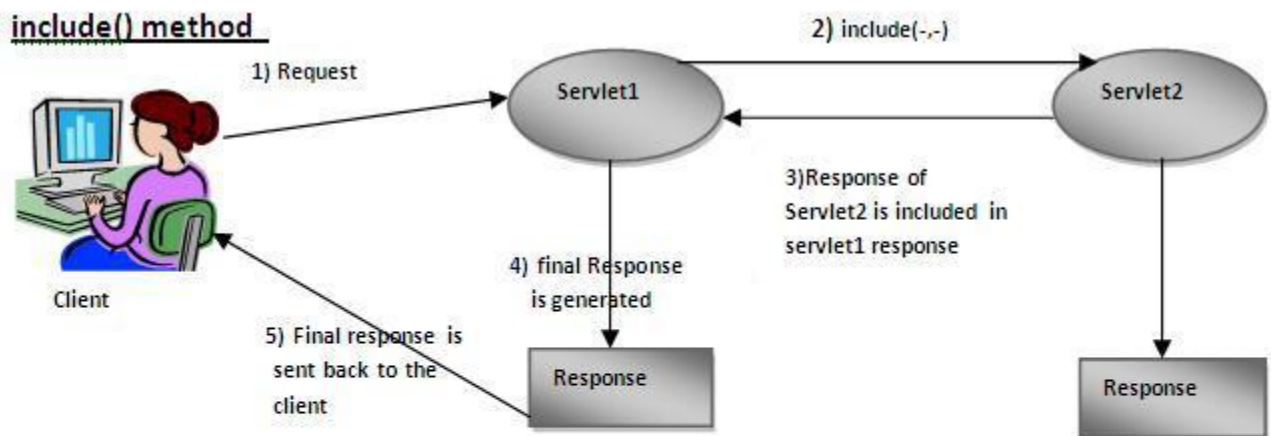
The RequestDispatcher interface provides two methods. They are:

1. **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.
2. **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() method:



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.



As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

How to get the object of RequestDispatcher

The `getRequestDispatcher()` method of `ServletRequest` interface returns the object of `RequestDispatcher`. Syntax:

Syntax of `getRequestDispatcher` method

```
public RequestDispatcher getRequestDispatcher(String resource);
```

Example of using `getRequestDispatcher` method

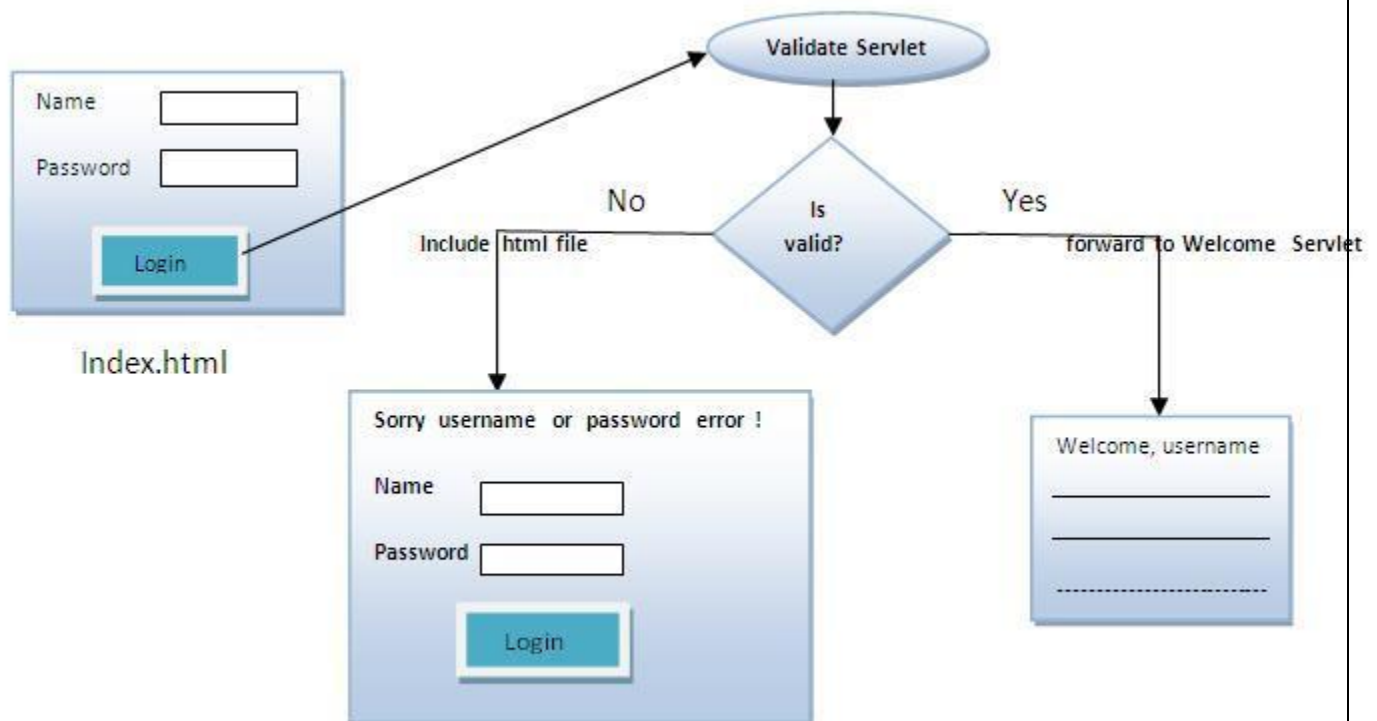
```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward
```

Example of RequestDispatcher interface

In this example, we are validating the password entered by the user. If password is servet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

- **index.html file:** for getting input from the user.
- **Login.java file:** a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class for displaying the welcome message.
- **web.xml file:** a deployment descriptor file that contains the information about the servlet.



index.html

```
<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>
```

Login.java

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

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);
        }
    }
}
```

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);
    }
}
```

```
}
```

web.xml

```
<web-app>  
<servlet>  
  <servlet-name>Login</servlet-name>  
  <servlet-class>Login</servlet-class>  
</servlet>  
<servlet>  
  <servlet-name>WelcomeServlet</servlet-name>  
  <servlet-class>WelcomeServlet</servlet-class>  
</servlet>  
  
<servlet-mapping>  
  <servlet-name>Login</servlet-name>  
  <url-pattern>/servlet1</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
  <servlet-name>WelcomeServlet</servlet-name>  
  <url-pattern>/servlet2</url-pattern>  
</servlet-mapping>  
  
<welcome-file-list>  
  <welcome-file>index.html</welcome-file>  
</welcome-file-list>  
</web-app>
```

SendRedirect in servlet

1. [sendRedirect method](#)
2. [Syntax of sendRedirect\(\) method](#)
3. [Example of RequestDispatcher interface](#)

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. So, it can work inside and outside the server.

Difference between forward() and sendRedirect() method

There are many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface. They are given below:

forward() method	sendRedirect() method
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.
Example: request.getRequestDispatcher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

Syntax of sendRedirect() method

```
public void sendRedirect(String URL)throws IOException;
```

Example of sendRedirect() method

```
response.sendRedirect("http://www.cmrct.org");
```

Full example of sendRedirect method in servlet

In this example, we are redirecting the request to the google server. Notice that sendRedirect method works at client side, that is why we can our request to anywhere. We can send our request within and outside the server.

DemoServlet.java

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

public class DemoServlet extends HttpServlet{
```

```

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

response.sendRedirect("http://www.google.com");

pw.close();
}}

```

Creating custom google search using sendRedirect

In this example, we are using sendRedirect method to send request to google server with the request data.

```

index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>

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

</body>
</html>

```

```

MySearcher.java
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);
    }
}

```

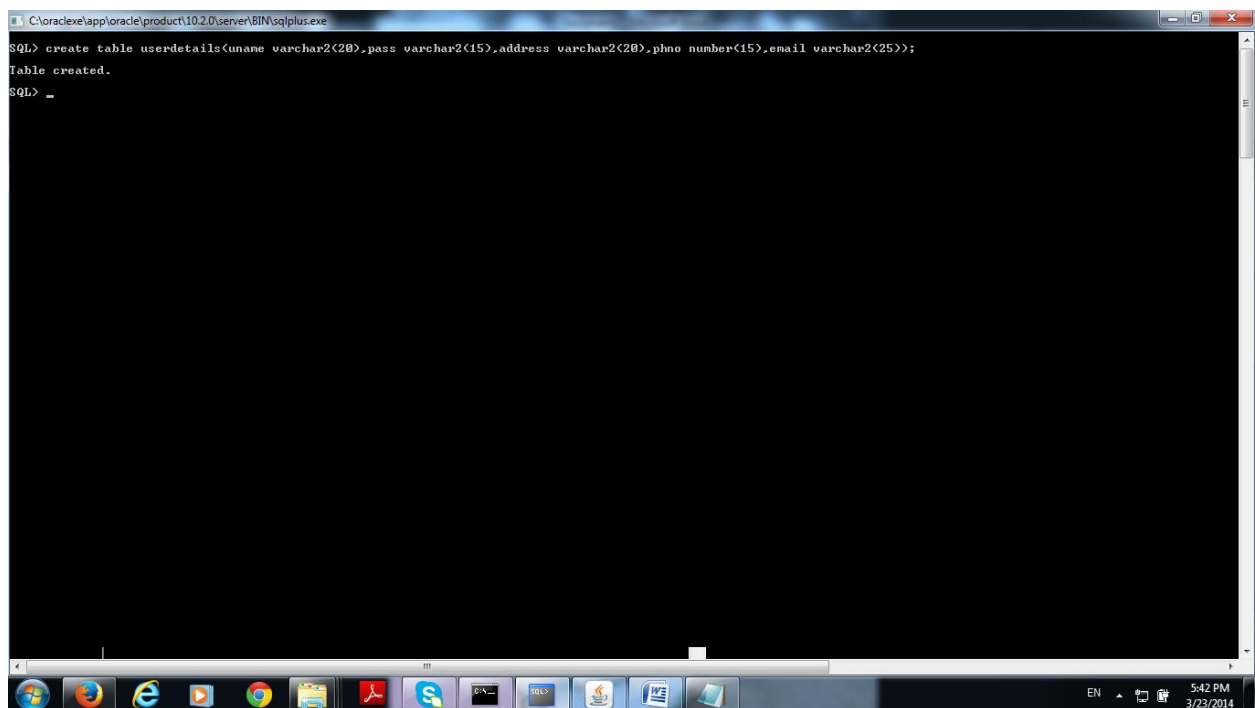

}

1. AIM: Write a Servlet for Registration to connect to database Insert the details of the users who register with the web site.
2. Write a Servlet for Login to connect to database using registration details

Crte the Table in Oracle Database

```
SQL> create table userdetails(uname varchar2(20),pass varchar2(15),address varchar2(20),phno number(15),email varchar2(25));
```

Table created.



The screenshot shows a Windows command prompt window titled "C:\oracle\app\oracle\product\10.2.0\server\BIN\sqlplus.exe". The command entered is: `SQL> create table userdetails(uname varchar2(20),pass varchar2(15),address varchar2(20),phno number(15),email varchar2(25));`. The output is: `Table created.` followed by a prompt `SQL> _`. The Windows taskbar at the bottom shows the system tray with the date 3/23/2014 and time 5:42 PM.

Working With Context Initialization Parameters

Let's create cmrsite application.

You need to perform the following broad level steps to create this application:

1. Create the Home.html file
2. Create the Login.html file
3. Create Register.html file

4. Create the RegistrationServlet Servlet
5. Create the LoginServlet Servlet
6. Create the Deployment descriptor file
7. Run the application

Crte the Home.html file

```
<!--Home.html-->

<html><body>

<center><h1>CMR COLLEGE OF ENGINEERING & TECHNOLOGY.</h1></center>

<table border="1" width="100%" height="100%">

<tr>

    <td width="15%" valign="top" align="center">

        <br/><a href="Login.html">Login</a><br/>

        <br/><a href="Register.html">Register</a><br/>

    </td>

    <td valign="top" align="center"><br/>

        Welcome to CMRCET

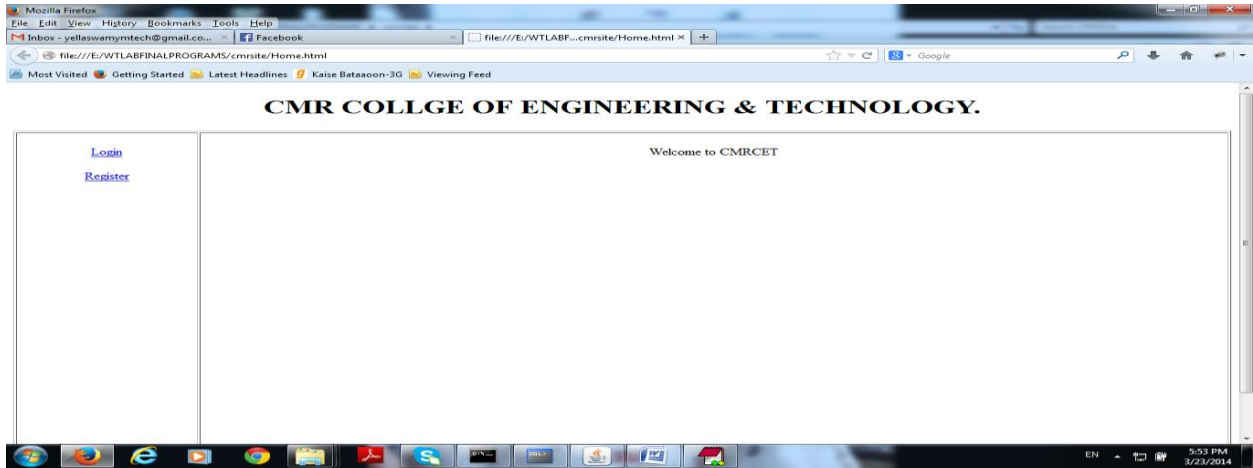
    </td>

</tr>

</table>

</body></html>
```

Output:



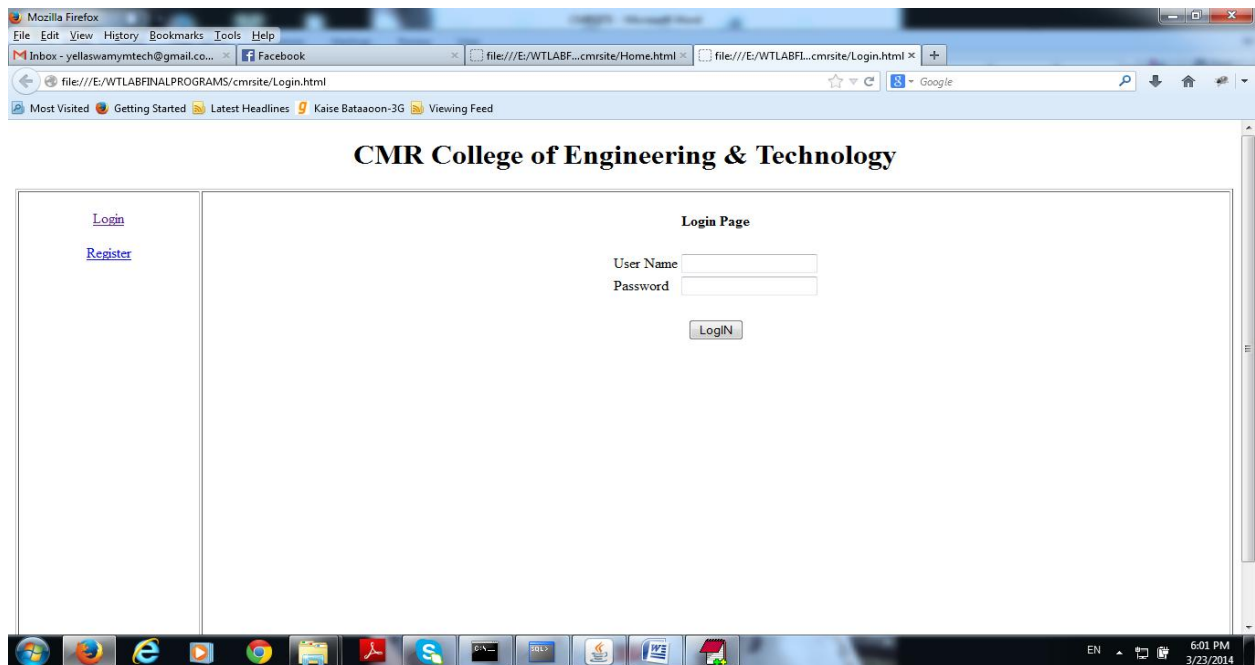
Creating the Login.html file

```

<!--Login.html-->
<html> <body>
<center><h1>CMR College of Engineering & Technology</h1></center>
<table border="1" width="100%" height="100%">
<tr>
<td width="15%" valign="top" align="center">
    <br/><a href="Login.html">Login</a><br/>
    <br/><a href="Register.html">Register</a><br/>
</td>
<td valign="top" align="center"><br/>
    <form action="login"><table>
<tr>
<td colspan="2" align="center"><b>Login Page</b></td>
</tr>
<tr>
<td colspan="2" align="center"><b>&nbsp;</b></td>
</tr>
<tr>
<td>User Name</td>
<td><input type="text" name="uname"/></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="pass"/></td>
</tr>
<tr>
<td>&nbsp;</td>
<td>&nbsp;</td>
</tr>
<tr>
<td colspan="2" align="center"><input type="submit" value="LogIN"/></td>
</tr>
</table></form>
</td>
</tr>
</table>
</body></html>

```

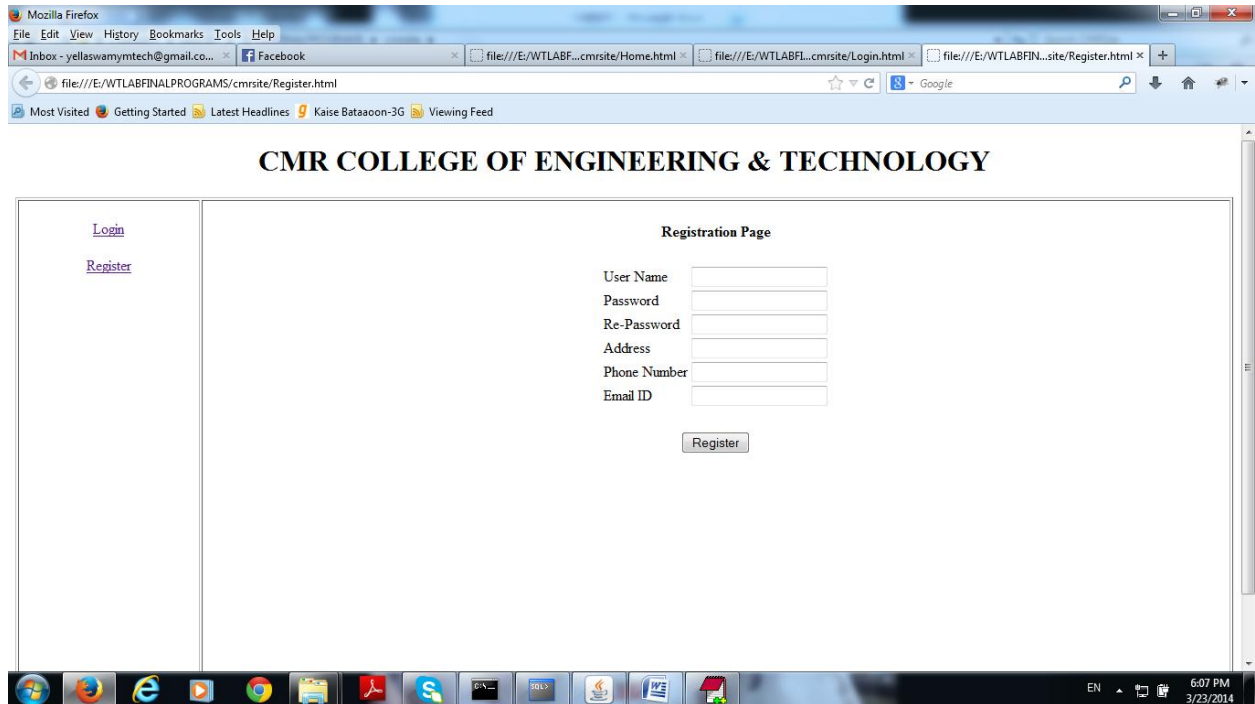
Output:



Creating the Register.html file

```
<!--Register.html-->
<html> <body>
<center><h1>CMR COLLEGE OF ENGINEERING & TECHNOLOGY</h1></center>
<table border="1" width="100%" height="100%">
<tr>
    <td width="15%" valign="top" align="center">
        <br/><a href="Login.html">Login</a><br/>
        <br/><a href="Register.html">Register</a><br/>
    </td>
    <td valign="top" align="center"><br/>
        <form action="register"><table>
            <tr>
                <td colspan="2" align="center"><b>Registration Page</b></td>
            </tr>
            <tr>
                <td colspan="2" align="center"><b>&nbsp;</b></td>
            </tr>
            <tr>
                <td>User Name</td>
                <td><input type="text" name="uname"/></td>
            </tr>
            <tr>
                <td>Password</td>
                <td><input type="password" name="pass"/></td>
            </tr>
            <tr>
                <td>Re-Password</td>
                <td><input type="password" name="repass"/></td>
            </tr>
            <tr>
                <td>Address</td>
                <td><input type="text" name="addr"/></td>
            </tr>
            <tr>
                <td>Phone Number</td>
                <td><input type="text" name="phno"/></td>
            </tr>
            <tr>
                <td>Email ID</td>
                <td><input type="text" name="email"/></td>
            </tr>
            <tr>
                <td>&nbsp;</td>
                <td>&nbsp;</td>
            </tr>
            <tr><td colspan="2" align="center"><input type="submit" value="Register"/></td>
            </tr>
        </table></form></td></tr></table></body></html>
```

Output:



Creating the RegistrationServlet Servlet

```
//RegistrationServlet.java
package com.yellaswamy.servlets;

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

/**
 * @author yellaswamy
 */
public class RegistrationServlet extends GenericServlet {

    private Connection con;
    private PreparedStatement ps=null;

    public void init() throws ServletException {
        System.out.println("In init");
        try {
            ServletContext ctxt=getServletContext();
```

```

        //Getting the Driver class name from context parameter
        String driverClassName=ctxt.getInitParameter("driverClassName");
        Class.forName(driverClassName);
        //Getting the JDBC URL from context parameter
        String url=ctxt.getInitParameter("url");

        //Getting the DB Username, password & sqlstatement from servlet init parameter
        String dbuser=getInitParameter("dbuser");
        String dbpass=getInitParameter("dbpass");
        String sqlst=getInitParameter("sqlstatement");

        con=DriverManager.getConnection(url, dbuser, dbpass);
        ps=con.prepareStatement(sqlst);
    }//try
    catch(Exception e){
        e.printStackTrace();
        throw new ServletException("Initialization failed, Unable to get DB
connection");
    }//catch
} //init

public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException
{

    System.out.println("In service");

    res.setContentType("text/html");
    PrintWriter out=res.getWriter();
    try {

        String uname=req.getParameter("uname");
        String pass= req.getParameter("pass");
        String repass= req.getParameter("repass");

        if (uname==null || uname.equals("")
            || pass==null || pass.equals("")
            || !pass.equals(repass)) {

            out.println("<html><body><center>");
            out.println("<li><i>Given details are not valid to register</i></li><br/>");
            out.println("<li><i>Please try again later</i>");
            out.println("</center></body></html>");
            return;
        } //if
    }
}

```



```
public void destroy ()
{
    System.out.println("In destroy");
    try {con.close();}
    catch(Exception e){}
} //destroy
} //class
```

Creating the LoginServlet Servlet

```
//LoginServlet
package com.yellaswamy.servlets;

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

/**
 * @author yellaswamy
 */
public class LoginServlet extends GenericServlet {

    private Connection con;
    private PreparedStatement ps=null;

    public void init() throws ServletException {
        System.out.println("In init");
        try {
            ServletContext ctxt=getServletContext();
            //Getting the Driver class name from context parameter
            String driverClassName=ctxt.getInitParameter("driverClassName");
            Class.forName(driverClassName);
            //Getting the JDBC URL from context parameter
            String url=ctxt.getInitParameter("url");

            //Getting the DB Username, password & sqlstatement from servlet init
parameter
            String dbuser=getInitParameter("dbuser");
            String dbpass=getInitParameter("dbpass");
            String sqlst=getInitParameter("sqlstatement");

            con=DriverManager.getConnection(url, dbuser, dbpass);
            ps=con.prepareStatement(sqlst);
        }//try
        catch(Exception e){
            e.printStackTrace();
            throw new ServletException("Initialization failed, Unable to get DB
connection");
        }//catch
    }//init
```

```

public void service (ServletRequest req, ServletResponse res) throws ServletException,
IOException {

    System.out.println("In service");

    res.setContentType("text/html");
    PrintWriter out=res.getWriter();
    try {

        String uname=req.getParameter("uname");
        String pass= req.getParameter("pass");

        if (uname==null || uname.equals("") || pass==null || pass.equals("")) {

            out.println("<html><body><center>");
            out.println("<li><i>User Name and Password cannot be
empty</i></li><br/>");
            out.println("<li><i>We cannot log you into your account at this time.
Please try again later</i>");
            out.println("</center></body></html>");
            return;
        }//if

        ps.setString(1,uname);
        ps.setString(2,pass);
        ResultSet rs=ps.executeQuery();

        if (rs.next()){
            out.println("<html><body>");
            out.println("<center><h1>CMR COLLEGE OF ENGINEERING &
Technology</h1></center>");
            out.println("<table border='1' width='100%' height='100%'>");
            out.println("<tr>");
            out.println("<td width='15%' valign='top' align='center'>");
            out.println("<br/><a href='Login.html'>Login</a><br/>");
            out.println("<br/><a href='Register.html'>Register</a>");
            out.println("</td>");
            out.println("<td valign='top' align='center'><br/>");
            out.println("<h3>Welcome, "+uname+"</h3><br/>");
            out.println("<h2>Enjoy browsing the Site</h2>");
            out.println("</td></tr></table>");
        }
    }
}

```

```

        out.println("</body></html>");
    }
    else{
        out.println("<html><body><center>");
        out.println("Given username and password are incorrect<br/>");
        out.println("<li><i>We cannot log you into your account at this time.
Please try again later</i>");
        out.println("</center></body></html>");
    }
} //try
catch(Exception e){
    out.println("<html><body><center>");
    out.println("<h2>Unable to the process the request try after some time</h2>");
    out.println("</center></body></html>");
} //catch
} //service

public void destroy () {
    System.out.println("In destroy");
    try {con.close();}
    catch(Exception e){}
} //destroy
} //class

```

Creating the Deployment Descriptor file(web.xml)

```
<?xml version="1.0"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-
app_2_3.dtd">
<web-app>

    <context-param>
        <param-name>driverClassName</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </context-param>
    <context-param>
        <param-name>url</param-name>
        <param-value>
            jdbc:oracle:thin:@localhost:1521:xe
        </param-value>
    </context-param>

    <servlet>
        <servlet-name>ls</servlet-name>
        <servlet-class>com.yellaswamy.servlets.LoginServlet</servlet-class>

        <init-param>
            <param-name>dbuser</param-name>
            <param-value>yellaswamy</param-value>
        </init-param>
        <init-param>
            <param-name>dbpass</param-name>
            <param-value>yellaswamy</param-value>
        </init-param>
        <init-param>
            <param-name>sqlstatement</param-name>
            <param-value>
                select * from userdetails where uname=? and pass=?
            </param-value>
        </init-param>
    </servlet>

    <servlet>
        <servlet-name>rs</servlet-name>
        <servlet-class>
```

```
        com.yellaswamy.servlets.RegistrationServlet
    </servlet-class>

    <init-param>
        <param-name>dbuser</param-name>
        <param-value>yellaswamy</param-value>
    </init-param>
    <init-param>
        <param-name>dbpass</param-name>
        <param-value>yellaswamy</param-value>
    </init-param>
    <init-param>
        <param-name>sqlstatement</param-name>
        <param-value>
            insert into userdetails values(?,?,?,?)
        </param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>ls</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>rs</servlet-name>
    <url-pattern>/register</url-pattern>
</servlet-mapping>

<welcome-file-list>
<welcome-file>Home.html</welcome-file>
</welcome-file-list>

</web-app>
```

Now arrange all the files in a directory structure ,as shown below

```
E:\WTLABFINALPROGRAMS\cmrsite>tree/f
```

```
Folder PATH listing for volume data
```

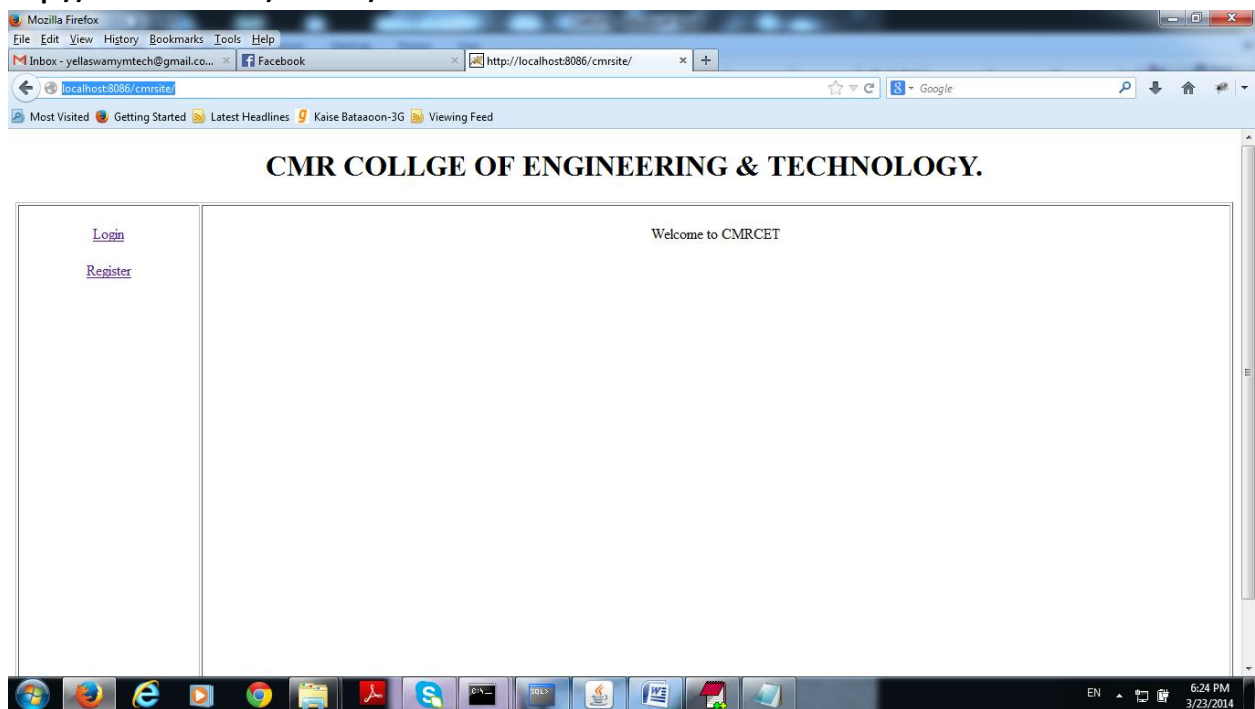
```
Volume serial number is 08FA-885D
```

```
E:.
```

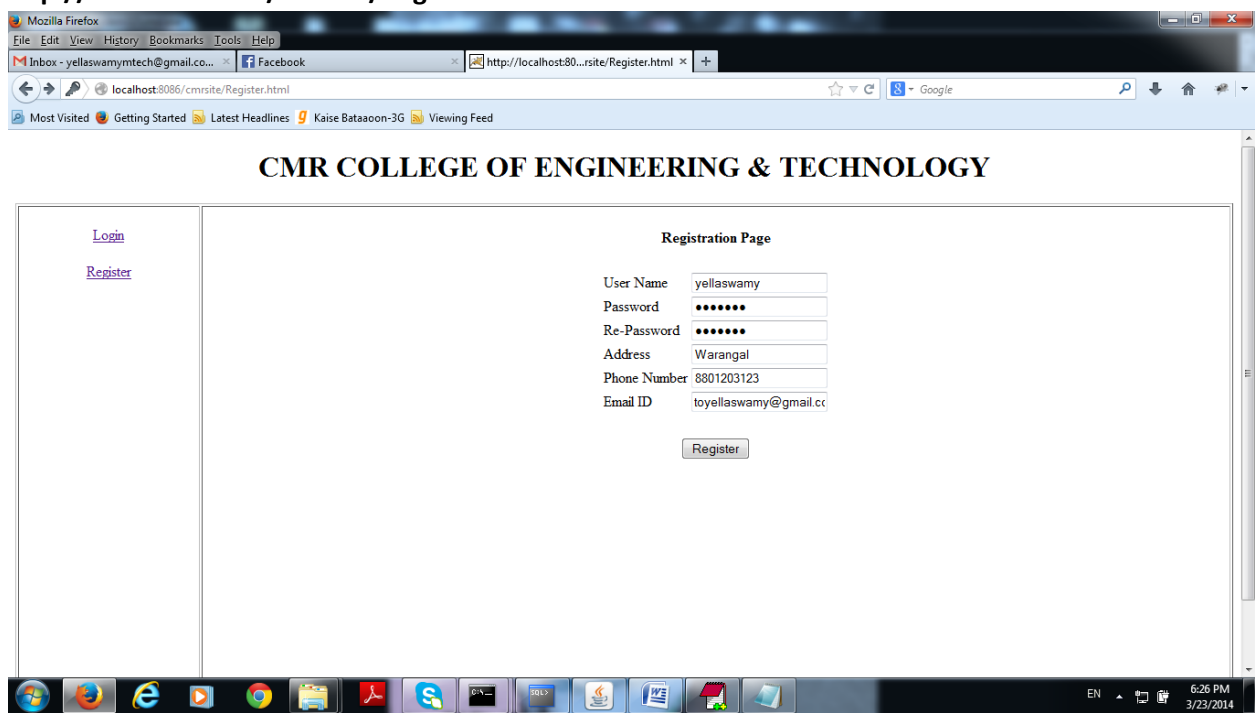
```
| CMRSITE.docx  
| Home.html  
| Login.html  
| Register.html  
|  
└─WEB-INF  
  | web.xml  
  |  
  └─classes  
    | | LoginServlet.java  
    | | RegistrationServlet.java  
    | |  
    └─com  
      └─yellaswamy  
        └─servlets  
          LoginServlet.class  
          RegistrationServlet.class  
|  
└─lib  
  classes12.jar  
  ojdbc14.jar
```

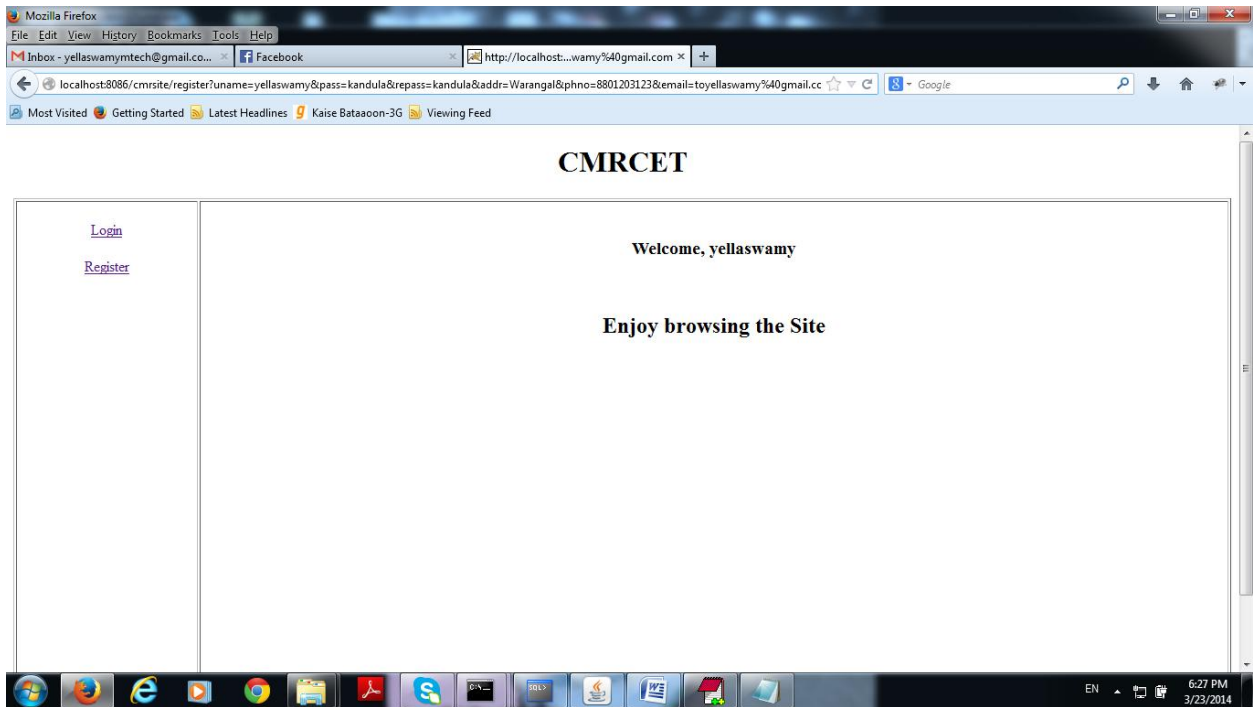
Run the Application

<http://localhost:8086/cmrsite/>



<http://localhost:8086/cmrsite/Register.html>





<http://localhost:8086/cmrsite/Login.html>

