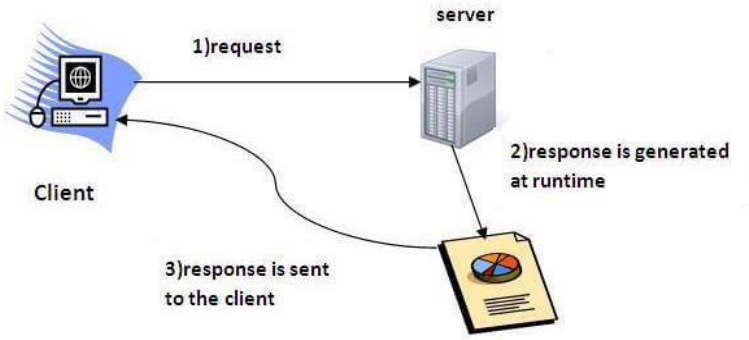
Servlets | Servlet Tutorial

- **Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).
- **Servlet** technology is robust and scalable because of java language.
- Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language.
- However, there were many disadvantages to this technology. We have discussed these disadvantages below.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

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

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

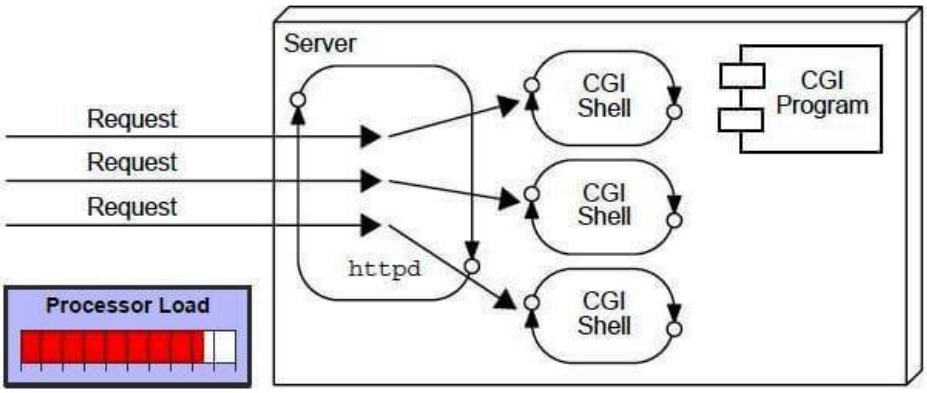


What is a web application?

- A web application is an application accessible from the web.
- A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript.
- The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request.
- For each request, it starts a new process.

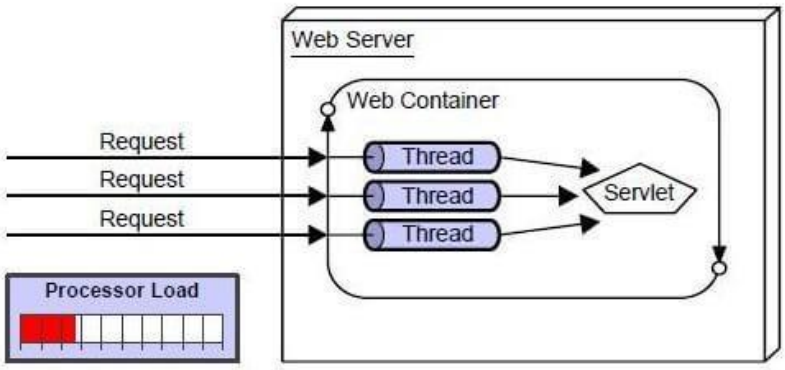


Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet



There are many advantages of Servlet over CGI.

- The web container creates threads for handling the multiple requests to the Servlet.
- Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low.
- The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Web Terminology - javatpoint

Web Terminology

Servlet Terminology	Description

Website: static vs dynamic	It is a collection of related web pages that may contain text, images, audio and video.
HTTP	It is the data communication protocol used to establish communication between client and server.
HTTP Requests	It is the request send by the computer to a web server that contains all sorts of potentially interesting information.
Get vs Post	It gives the difference between GET and POST request.
Container	It is used in java for dynamically generating the web pages on the server side.
Server: Web vs Application	It is used to manage the network resources and for running the program or software that provides services.
Content Type	It is HTTP header that provides the description about what are you sending to the browser.

Website

- Website is a collection of related web pages that may contain text, images, audio and video.
- The first page of a website is called home page.
- Each website has specific internet address (URL) that you need to enter in your browser to access a website.
- Website is hosted on one or more servers and can be accessed by visiting its homepage using a computer network.
- A website is managed by its owner that can be an individual, company or an organization.

A website can be of two types:

- Static website
- Dynamic website

Static website

- Static website is the basic type of website that is easy to create.
- You don't need the knowledge of web programming and database design to create a static website. Its web pages are coded in HTML.
- The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.

Dynamic website

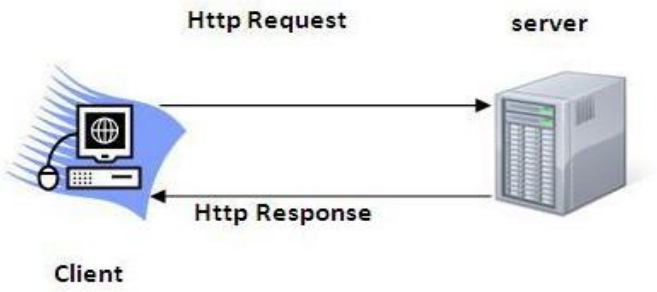
- Dynamic website is a collection of dynamic web pages whose content changes dynamically.
- It accesses content from a database or Content Management System (CMS).
- Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.
- Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content.
- Client side scripting generates content at the client computer on the basis of user input.
- The web browser downloads the web page from the server and processes the code within the page to render information to the user.
- In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.

Static vs Dynamic website

Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It sends exactly the same response for every request.	It may generate different HTML for each of the request.
The content is only changed when someone publishes and updates the file (sends it to the web server).	The page contains "server-side" code which allows the server to generate the unique content when the page is loaded.
Flexibility is the main advantage of static website.	Content Management System (CMS) is the main advantage of dynamic website.

HTTP (Hyper Text Transfer Protocol)

- The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems.
- It is the data communication protocol used to establish communication between client and server.
- HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP80.
- It provides the standardized way for computers to communicate with each other.



The Basic Characteristics of HTTP (Hyper Text Transfer Protocol):

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

The Basic Features of HTTP (Hyper Text Transfer Protocol):

- There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

HTTP is media independent: It specifies that any type of media content can be sent by HTTP as long as both the server and the client can handle the data content.

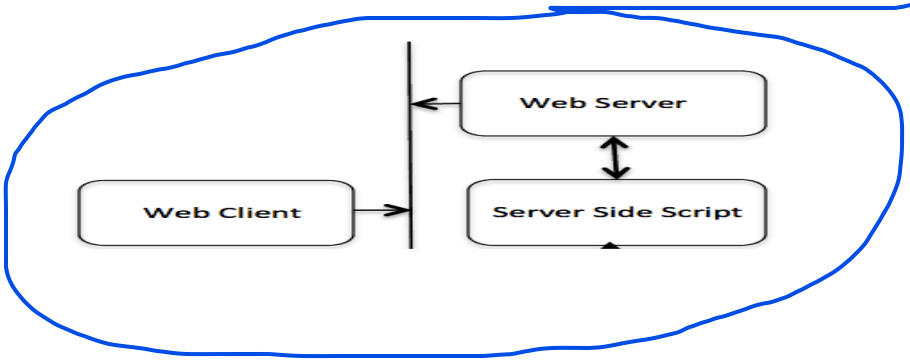
HTTP is connectionless: It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.

HTTP is stateless: The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor

the server can retain the information about different request across the web pages.

The Basic Architecture of HTTP (Hyper Text Transfer Protocol):

The below diagram represents the basic architecture of web application and depicts where HTTP stands:



HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server

HTTP Requests

- The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.
- The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

The HTTP request methods are:

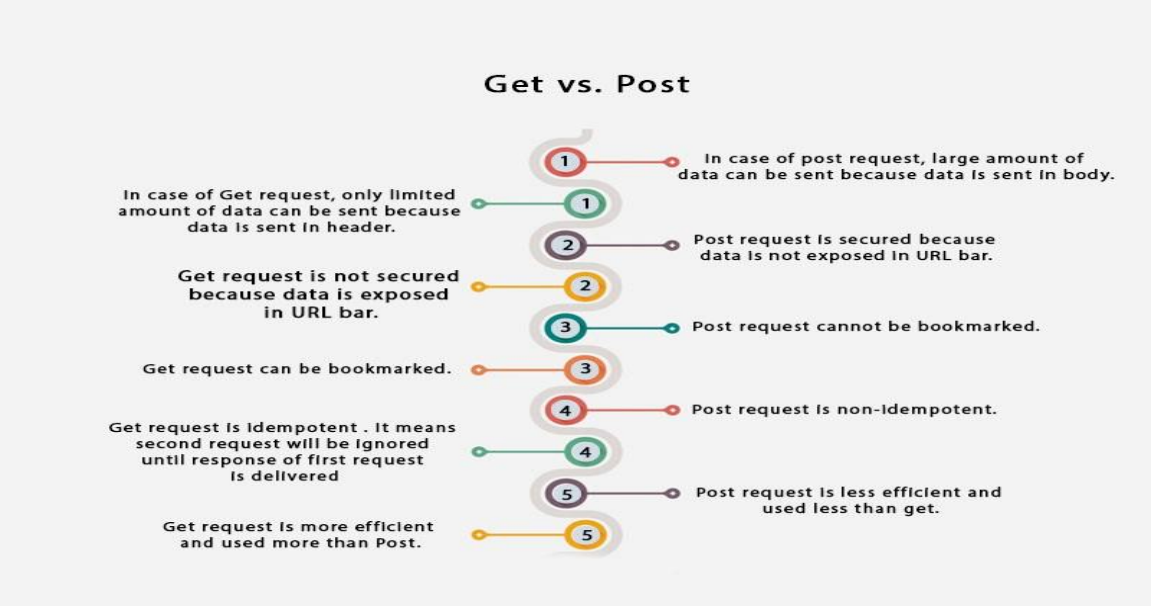
HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks 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

Get vs. 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.



GET and POST

Two common methods for the request-response between a server and client are:

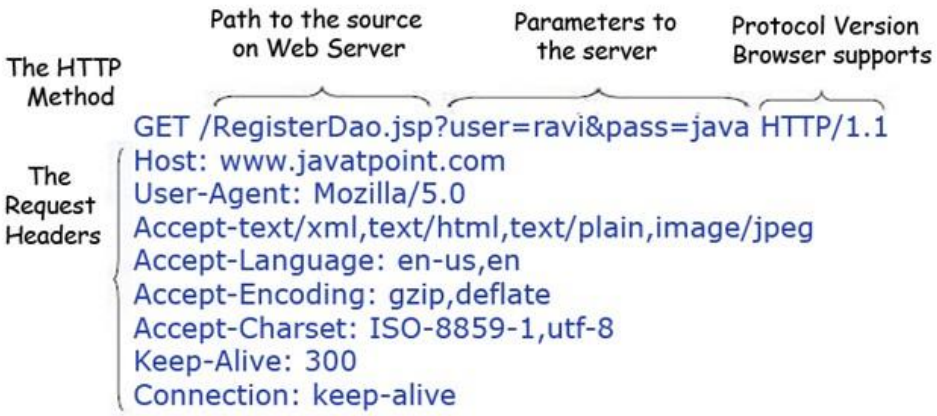
- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

Anatomy of Get Request

The query string (name/value pairs) is sent inside the URL of a GET request:

```
GET/RegisterDao.jsp?name1=value1&name2=value2
```

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 information is sent to the server.



Some other features of GET requests are:

- It remains in the browser history
- It can be bookmarked
- It can be cached
- It have length restrictions
- It should never be used when dealing with sensitive data

It should only be used for retrieving the data

Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a POST request:

```
POST/RegisterDao.jsp HTTP/1.1
Host: www. javatpoint.com
name1=value1&name2=value2
```

As we know, in case of post request original data is sent in message body.

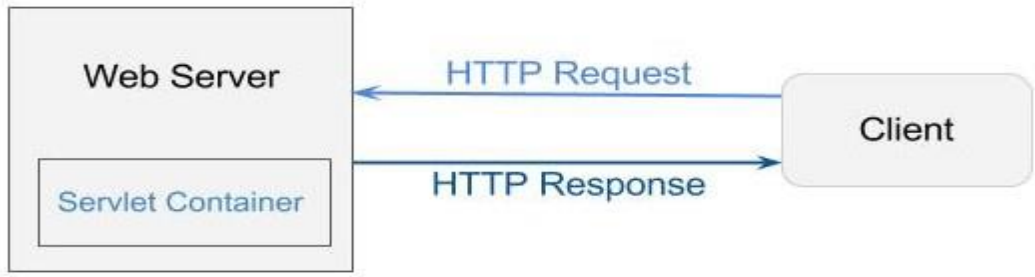
Let's see how information is passed to the server in case of post request.

Some other features of POST requests are:

- This requests cannot be bookmarked
- This requests have no restrictions on length of data
- This requests are never cached
- This requests do not retain in the browser history

Servlet Container

- It provides the runtime environment for JavaEE (j2ee) applications.
- The client/user can request only a static WebPages from the server.
- If the user wants to read the web pages as per input then the servlet container is used in java.
- The servlet container is the part of web server which can be run in a separate process.
- We can classify the servlet container states in three types:



Servlet Container States

The servlet container is the part of web server which can be run in a separate process. We can classify the servlet container states in three types:

- **Standalone:** It is typical Java-based servers in which the servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself
- **In-process:** It is separated from the web server, because a different program runs within the address space of the main server as a plug-in. For example:- Tomcat running inside the JBoss.
- **Out-of-process:** The web server and servlet container are different programs which are run in a different process. For performing the communications between them, web server uses the plug-in provided by the servlet container.

The Servlet Container performs many operations that are given below:

- Life Cycle Management
- Multithreaded support
- Object Pooling

Security etc.

Server: Web vs. Application

- Server is a device or a computer program that accepts and responds to the request made by other program, known as client.
- It is used to manage the network resources and for running the 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.

- It is a computer where the web content can be stored. In general web server can be used to host the web sites but there also used some other web servers also such as FTP, email, storage, gaming etc.

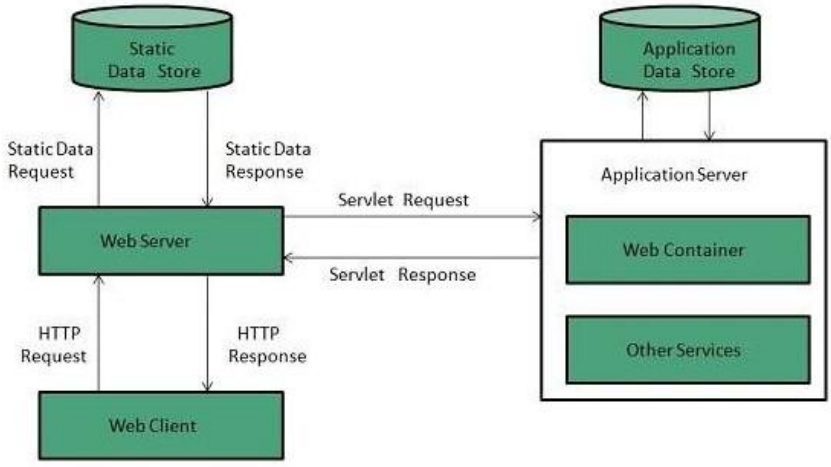
Examples of Web Servers are: **Apache Tomcat** and **Resin**.

Web Server Working

It can respond to the client request in either of the following two possible ways:

- Generating response by using the script and communicating with database.
- Sending file to the client associated with the requested URL.

The block diagram representation of Web Server is shown below:



Important points

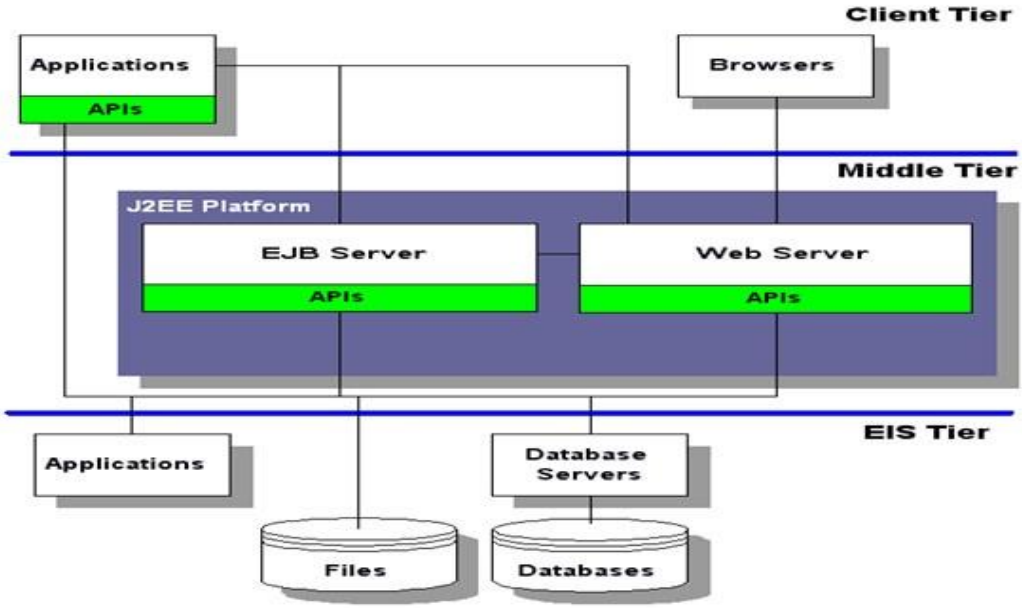
- If the requested web page at the client side is not found, then web server will sends the HTTP response: Error 404 Not found.
- When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
- If the client requests some other resources then web server will contact to application server and data is store for constructing the HTTP response.

Application Server

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

- It is a component based product that lies in the middle-tier of a server centric architecture.
- It provides the middleware services for state maintenance and security, along with persistence and data access.
- It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.

The block diagram representation of Application Server is shown below:



The 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.
- MIME is an internet standard that is used for extending the limited capabilities of email by allowing the insertion of sounds, images and text in a message.

The features provided by MIME to the email services are as given below:

- It supports the non-ASCII characters
- It supports the multiple attachments in a single messag
- It supports the attachment which contains executable audio, images and video files etc.
- It supports the unlimited message length.

List of Content Types

There are many content types. The commonly used content types are given below:

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- images/png
- images/gif

audio/mp3

video/mp4

video/quicktime etc.

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)

Servlet Interface

Servlet interface provides commonbehaviorto all the servlets.Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet Example by implementing Servlet interface

Let's see the simple example of servlet by implementing the servlet interface.

It will be better if you learn it after visiting steps to create a servlet.

File: First.java

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

public class First implements Servlet{
ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("servlet is initialized");
}

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

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");

}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}

}
```

GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.

11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

Servlet Example by inheriting the GenericServlet class

Let's see the simple example of servlet by inheriting the GenericServlet class.

> **It will be better if you learn it after visiting steps to create a servlet.**

File: First.java

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

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

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");

}
}
```

download this exampl e

HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

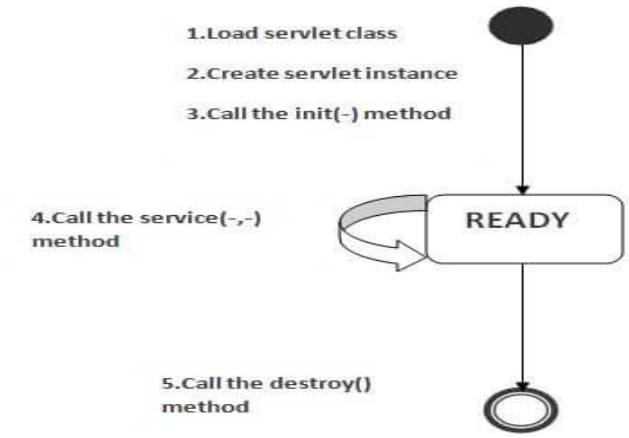
1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.

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

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end.

The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException
```

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

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

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()
```

Steps to create a servlet example

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:

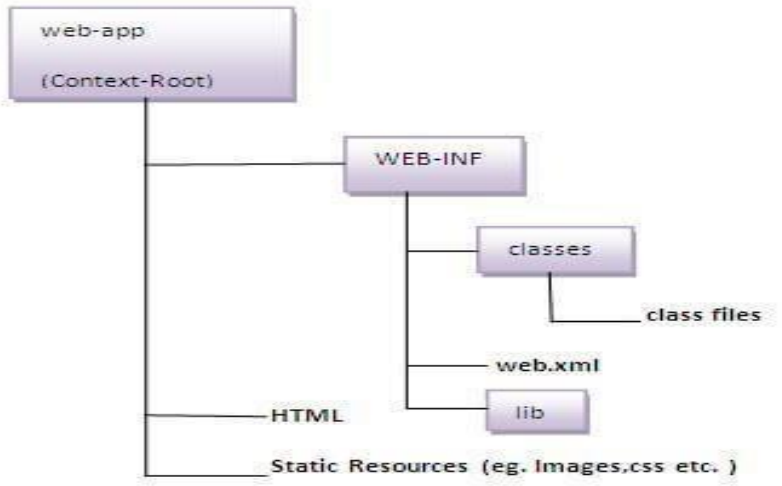
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

download this example of servlet download example of servlet by extending GenericServlet download example of servlet by implementing Servlet interface

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.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

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.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close();//closing the stream
}}
```

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.

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

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet 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 file

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

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).

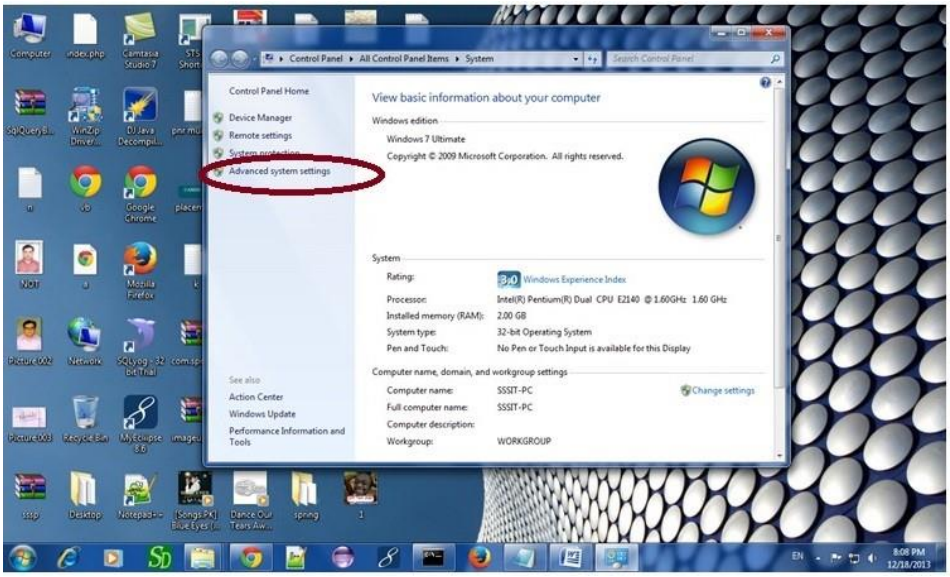
1) How to set JAVA_HOME in environment variable?

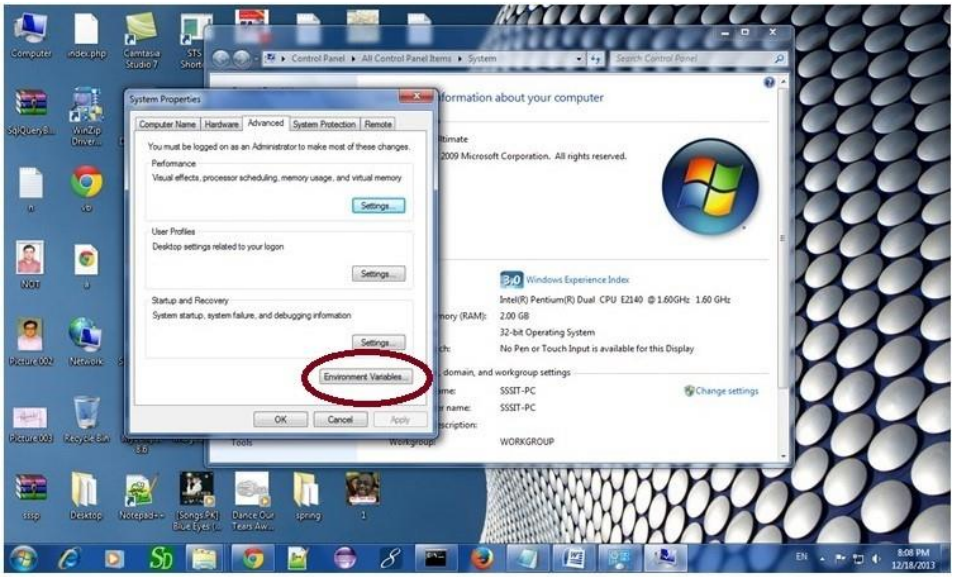
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

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:

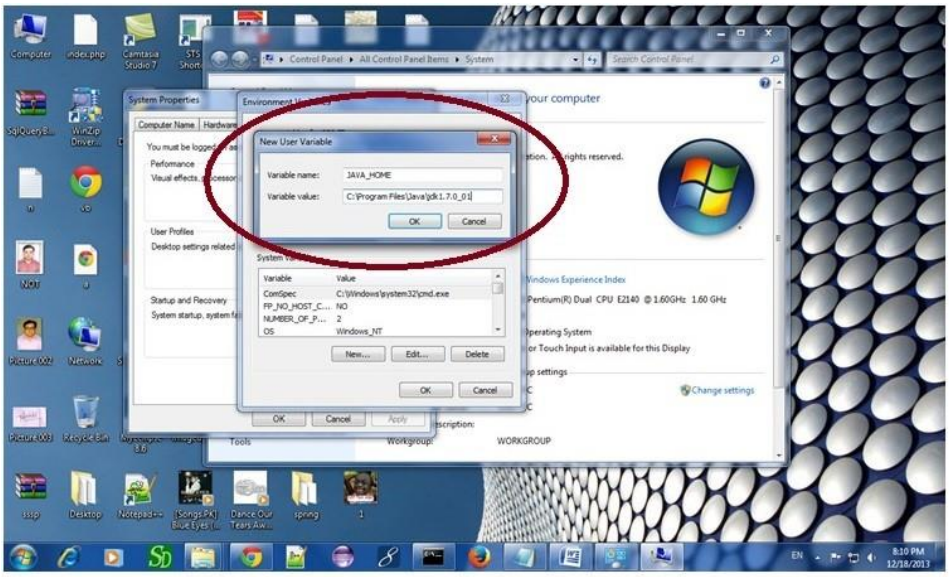
Click on advanced system settings tab then environment variables:





Click on the new tab of user variable or system variable:

Write JAVA_HOME in variable name and paste the path of jdk folder in variable value:



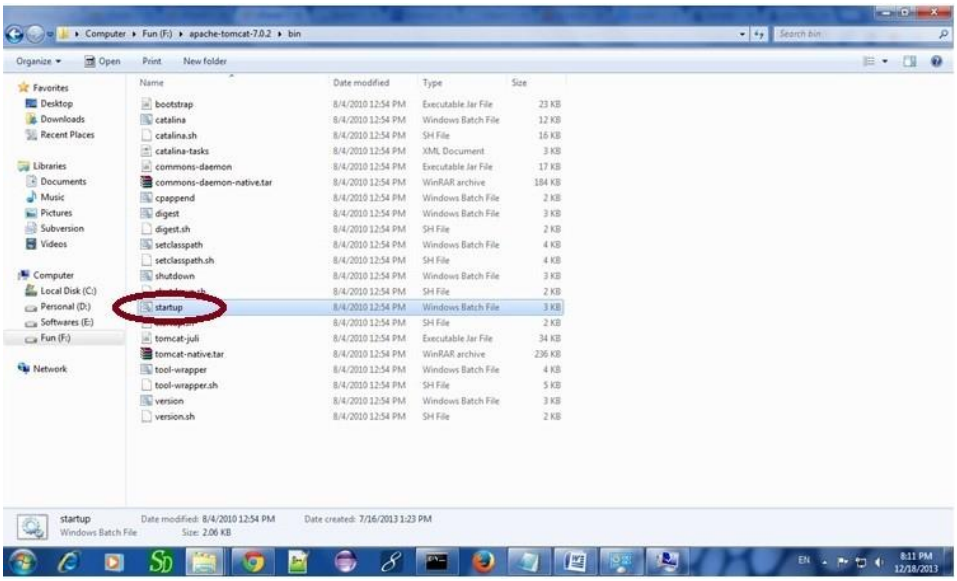
There must not be semicolon (;) at the end of the path.

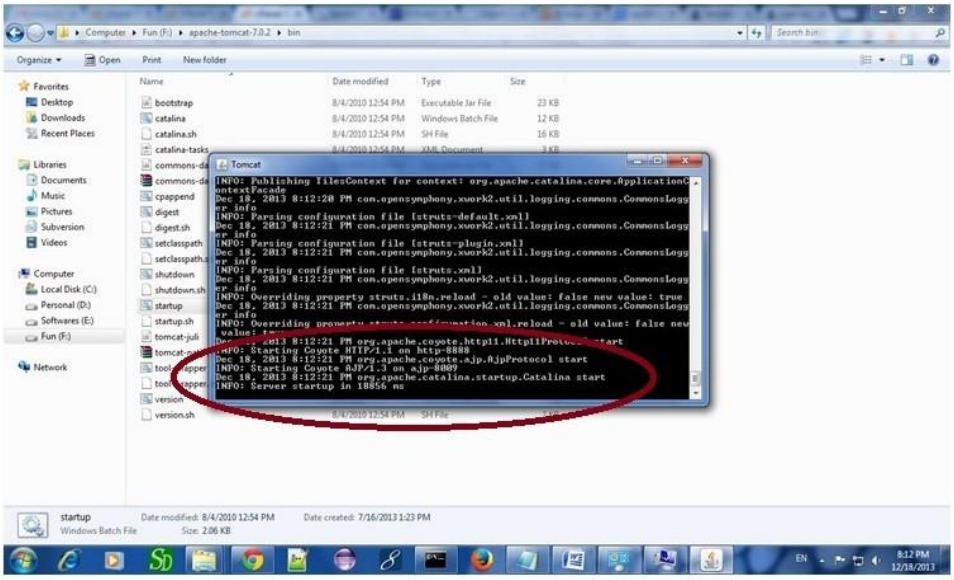
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

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.





Now server is started successfully.

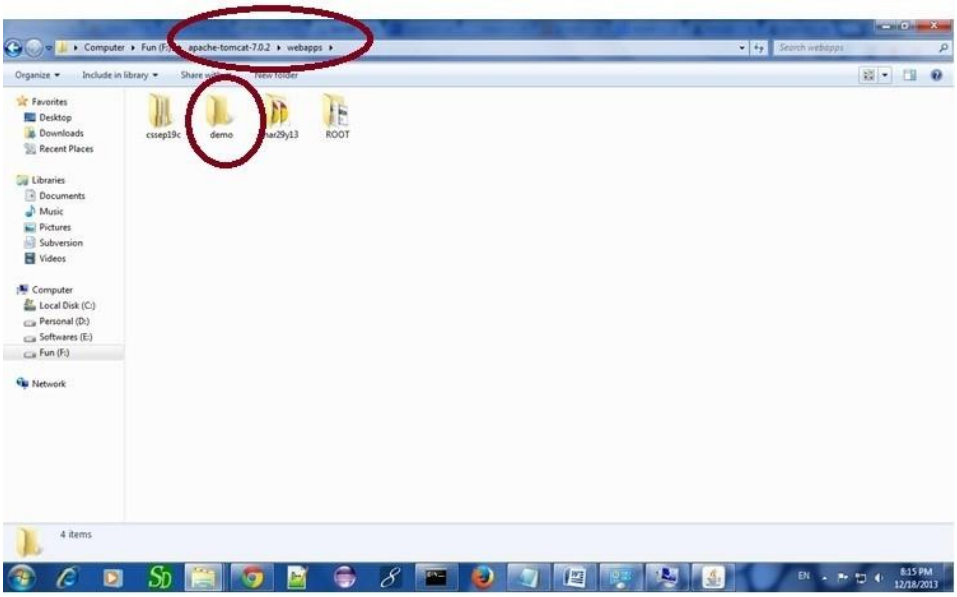
2) How to change port number of apache tomcat

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

5) How to deploy the servlet project

Copy the project and paste it in the webapps folder under apache tomcat.



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

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

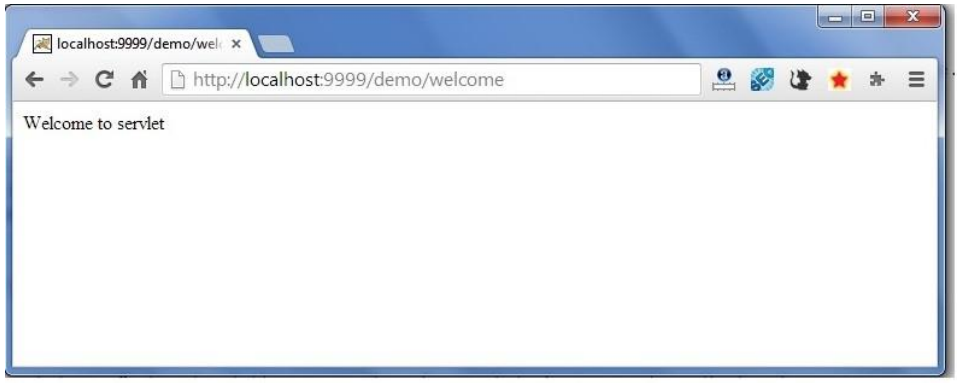
```
projectfolder> jar cvf myproject.war *
```

Creating war file has an advantage that moving the project from one location to another takes less time.

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

```
http://localhost:9999/demo/welcome
```



How Servlet works?

It is important to learn how servlet works for understanding the servlet well. Here, we are going to get the internal detail about the first servlet program.

The server checks if the servlet is requested **for the first time**.

If yes, web container does the following tasks:

- loads the servlet class.
- instantiates the servlet class. calls the init

method passing the ServletConfig object **else** calls

the service method passing request and response

objects

The web container calls the destroy method when it needs to remove the servlet such as at time of stopping server or undeploying the project.

How web container handles the servlet request?

The web container is responsible to handle the request. Let's see how it handles the request.

- maps the request with the servlet in the
- web.xml file. creates request and response
- objects for this request calls the service
- method on the thread

The public service method internally calls the protected service method

The protected service method calls the doGet method depending on the type of request.

The doGet method generates the response and it is passed to the client.

After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

What is written inside the public service method?

The public service method converts the ServletRequest object into the HttpServletRequest type and ServletResponse object into the HttpServletResponse type. Then, calls the service method passing these objects. Let's see the internal code:

```
public void service(ServletRequest req, ServletResponse res)
      throws ServletException, IOException
  {
     HttpServletRequest request;
     HttpServletResponse response;
     try
     {
        request = (HttpServletRequest)req;
        response = (HttpServletResponse)res;
     }
     catch(ClassCastException e)
     {
        throw new ServletException("non-HTTP request or response");
     }
     service(request, response);
  }
```

What is written inside the protected service method?

The protected service method checks the type of request, if request type is get, it calls doGet method, if request type is post, it calls doPost method, so on. Let's see the internal code:

```
protected void service(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
    {
        String method = req.getMethod();
        if(method.equals("GET"))
        {
            long lastModified = getLastModified(req);
            if(lastModified == -1L)
            {
                doGet(req, resp);
            }
    ....
    //rest of the code

        }
    }
```

War File

A **war (web archive) File** contains files of a web project. It may have servlet, xml, jsp, image, html, css, js etc. files.

Here, we will discuss what is war file, how to create war file, how to deploy war file and how to extract war file.

What is war file?

web archive (war) file contains all the contents of a web application. It reduces the time duration for transferring file.

Advantage of war file

saves time: The war file combines all the files into a single unit. So it takes less time while transferring file from client to server.

How to create war file?

To create war file, you need to use **jar tool** of JDK. You need to use **-c** switch of jar, to create the war file.

Go inside the project directory of your project (outside the WEB-INF), then write the following command:

```
jar -cvf projectname.war *
```

Here, **-c** is used to create file, **-v** to generate the verbose output and **-f** to specify the arhive file name.

The *** (asterisk) symbol** signifies that all the files of this directory (including sub directory).

How to deploy the war file?

There are two ways to deploy the war file.

1. By server console panel
2. By manually having the war file in specific folder of server.

If you want to deploy the war file in **apache tomcat** server manually, go to the **webapps** directory of apache tomcat and paste the war file here.

Now, you are able to access the web project through browser.

> Note: server will extract the war file internally.

How to extract war file manually?

To extract the war file, you need to use **-x switch** of **jar tool** of JDK. Let's see the command to extract the war file.

```
jar -xvf projectname.war
```

Creating Servlet Example in Eclipse

Eclipse is an open-source ide for developing JavaSE and JavaEE (J2EE) applications. You can download the eclipse ide from the eclipse website http://www.eclipse.org/downloads/.

You need to download the eclipse ide for JavaEE developers.

Creating **servlet example in eclipse ide**, saves a lot of work to be done. It is easy and simple to create a servlet example. Let's see the steps, you need to follow to create the first servlet example.

- Create a Dynamic web
- project create a
- servlet add servlet-

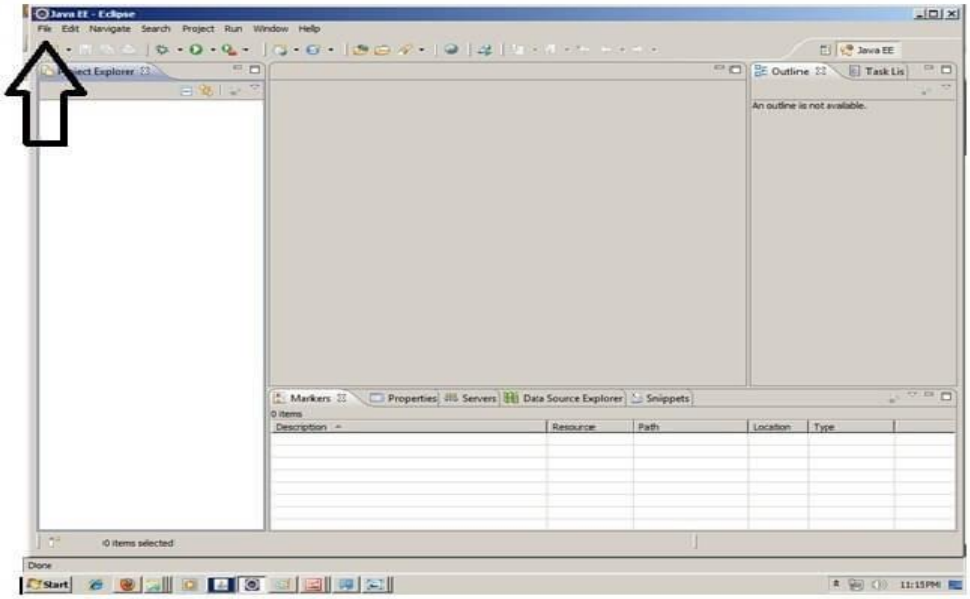
api.jar file

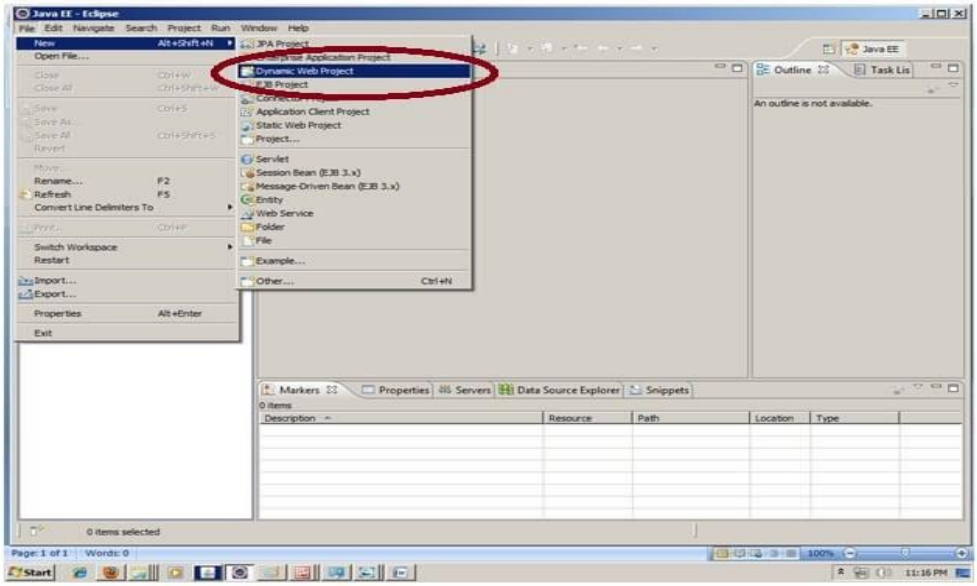
Run the servlet download this example

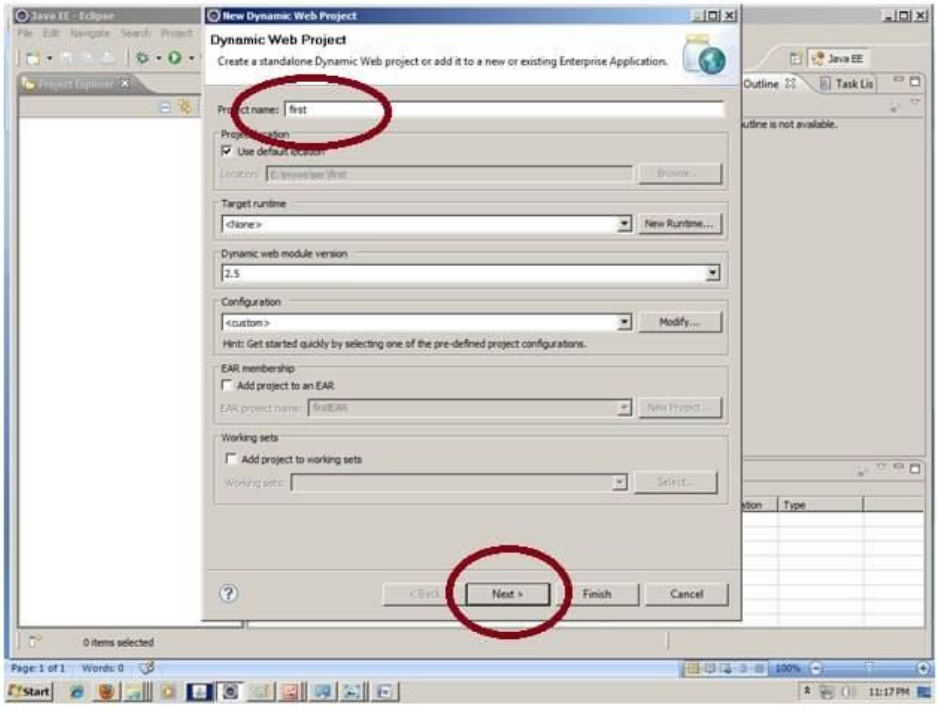
(developed in eclipse)

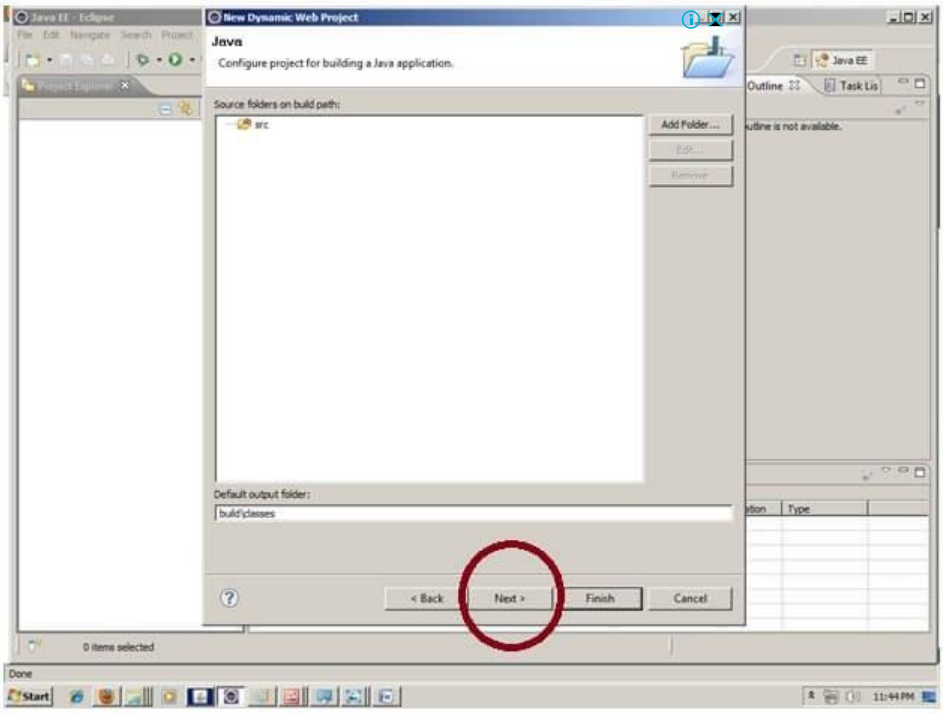
1) Create the dynamic web project:

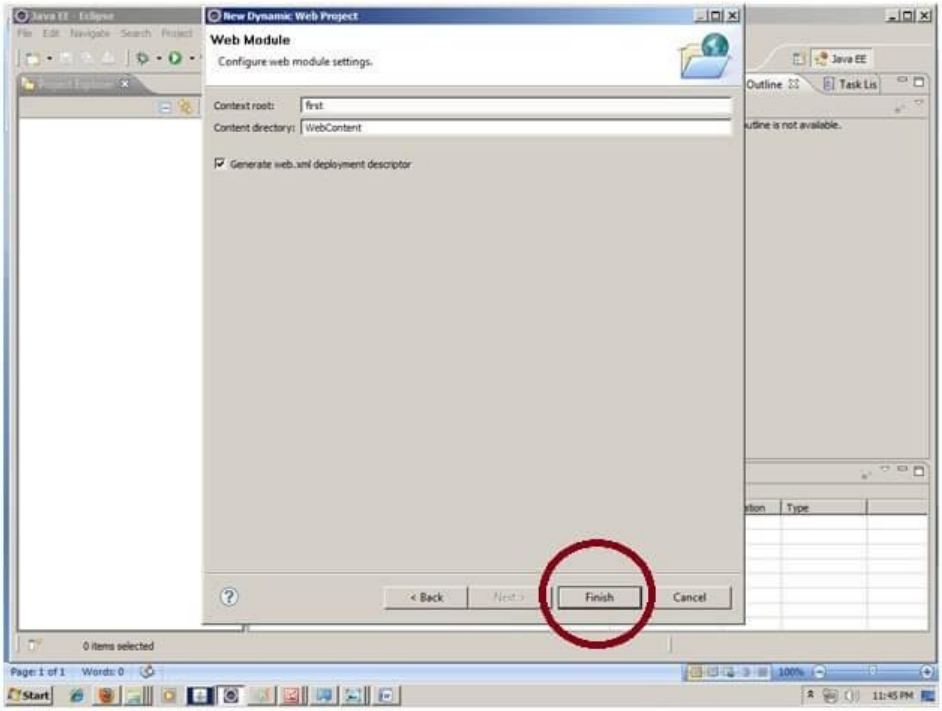
For creating a dynamic web project **click on File Menu -> New -> Project..-> Web -> dynamic web project -> write your project name e.g. first -> Finish**.

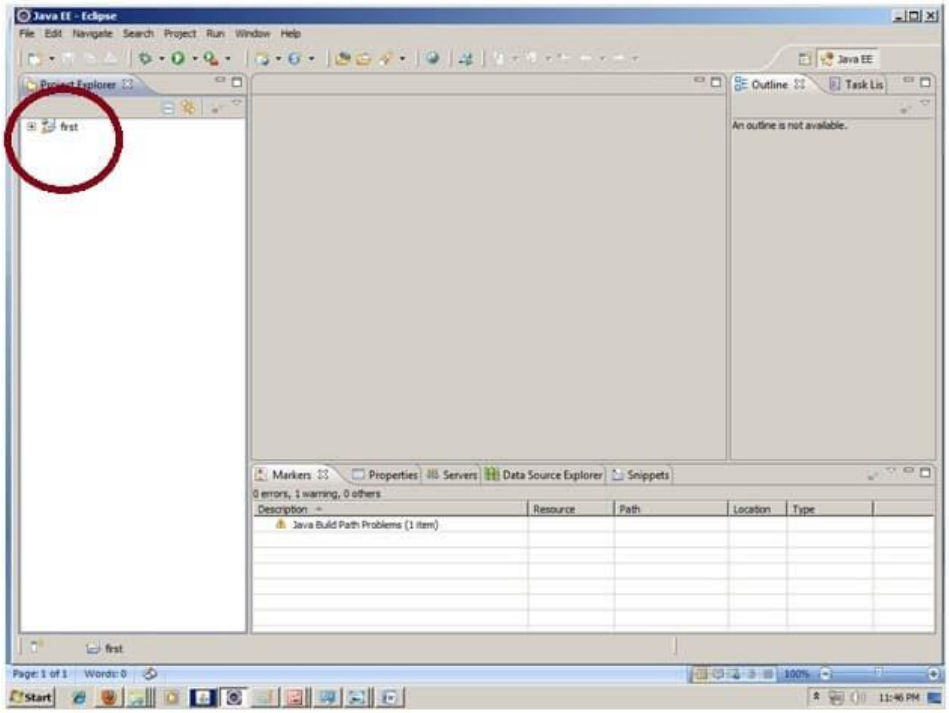

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Description
Resource
Path
Location
Type
Done
Start


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
JPA Project
Dynamic Web Project
EJB Project
Application Client Project
Static Web Project
Project...
Servlet
Session Bean (EJB 3.x)
Message-Driven Bean (EJB 3.x)
Entity
Web Service
Folder
File
Example...
Other... Ctrl+N
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
Description
Resource
Path
Location
Type
0 items selected
Page: 1 of 1
Words: 0
100%
Start


New Dynamic Web Project
Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.
Project name: first
Use default location
Target runtime
<none>
New Runtime...
Dynamic web module version
2.5
Configuration
<custom>
Modify...
Hint: Get started quickly by selecting one of the pre-defined project configurations.
EAR membership
Add project to an EAR
Working sets
Add project to working sets
< Back
Next >
Finish
Cancel
0 items selected

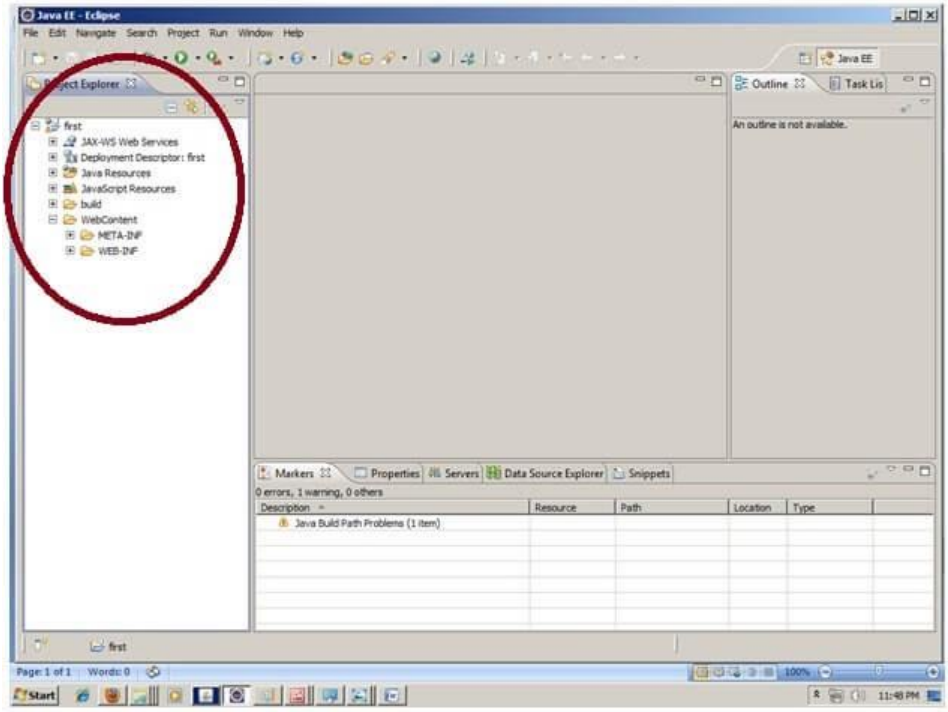

New Dynamic Web Project
Java
Configure project for building a Java application.
Source folders on build path:
src
Add Folder...
Default output folder:
build\classes
< Back
Next >
Finish
Cancel
0 items selected
Done

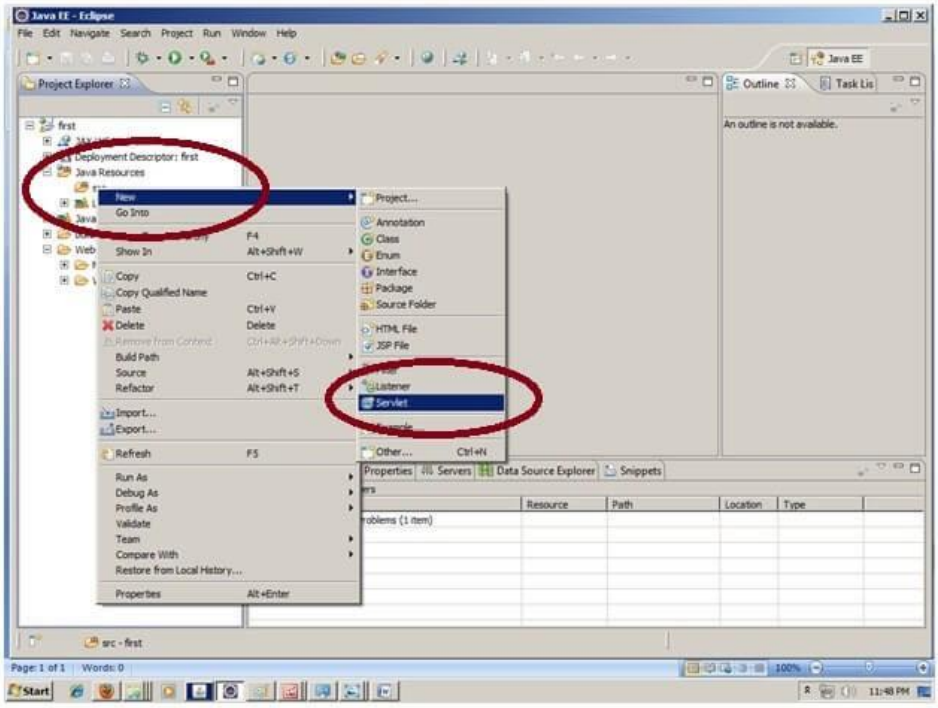


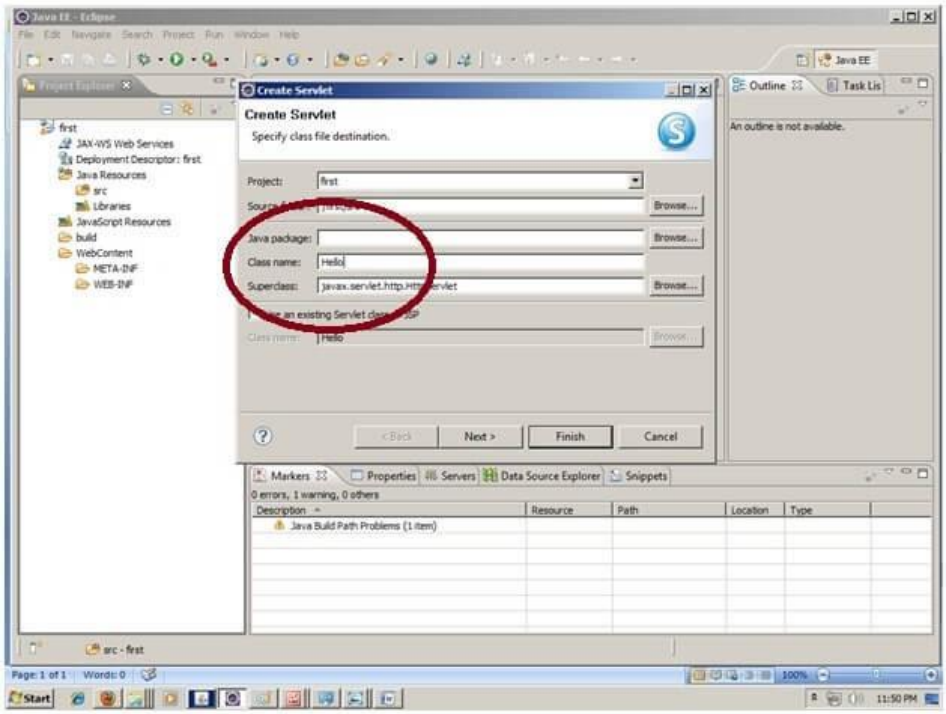


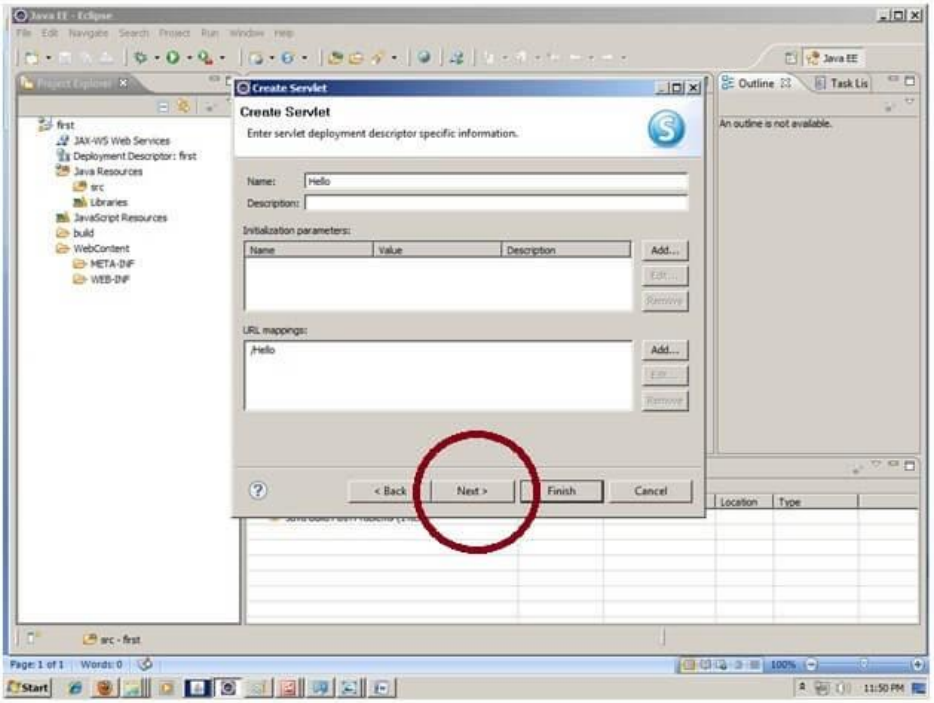
2) Create the servlet in eclipse IDE:

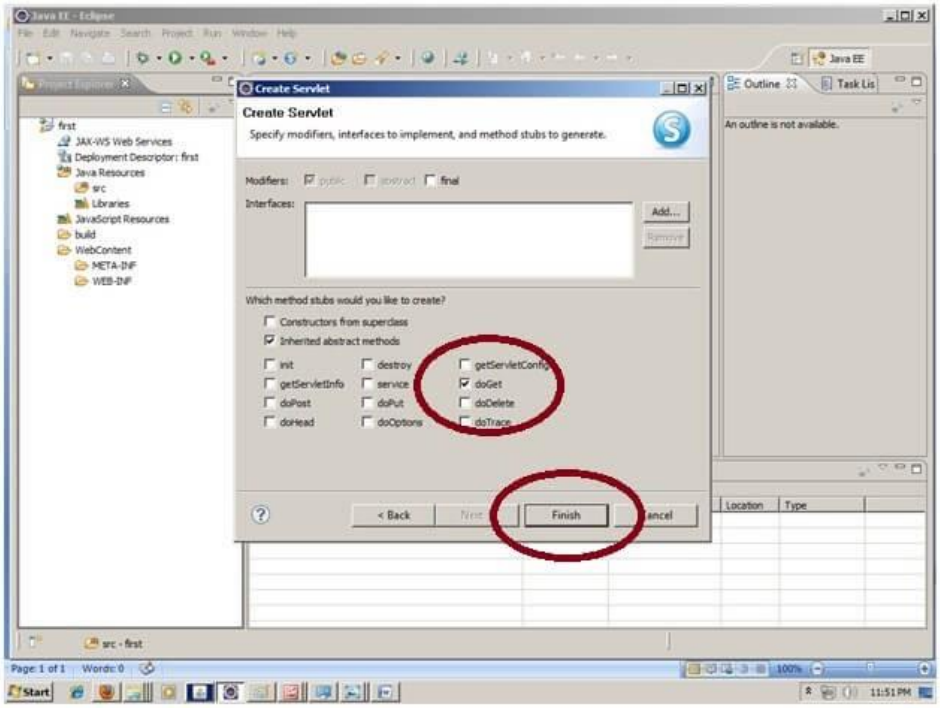
For creating a servlet, **explore the project by clicking the + icon -> explore the Java Resources -> right click on src -> New -> servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish**.

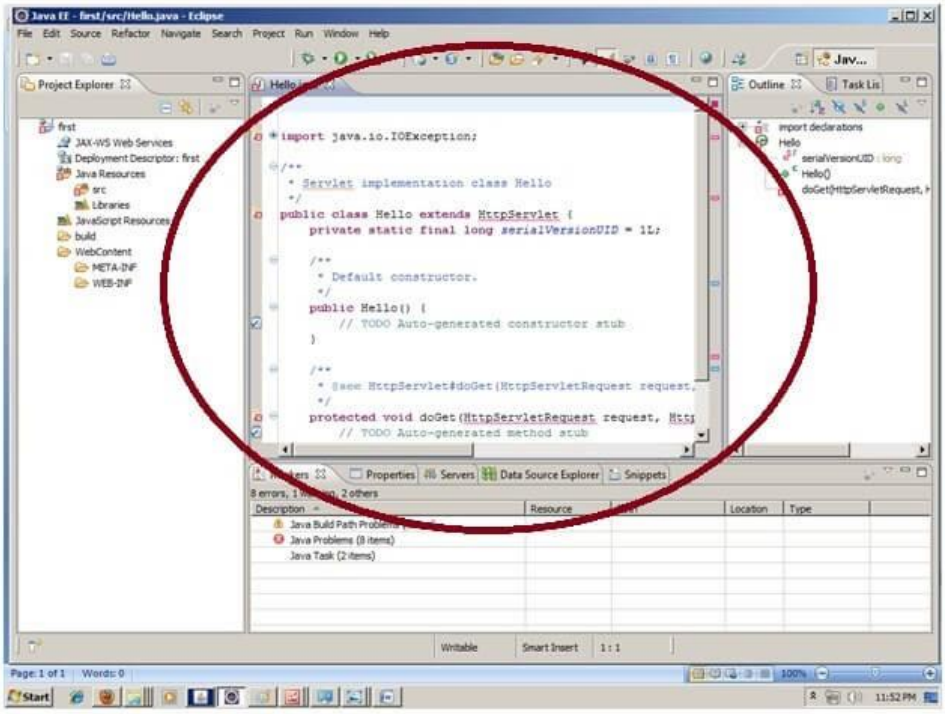

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
Task List
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Java Build Path Problems (1 item)


Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
Outline
Task List
An outline is not available.
New
Go Into
Show In
Copy
Copy Qualified Name
Paste
Delete
Build Path
Source
Refactor
Import...
Export...
Refresh
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
Properties
Project...
Annotation
Class
Enum
Interface
Package
Source Folder
HTML File
JSP File
Listener
Servlet
Other...
Markers
Properties
Servers
Data Source Explorer
Snippets
Resource
Path
Location
Type
src - first


Create Servlet
Create Servlet
Specify class file destination.
Project:
first
Java package:
Class name:
Hello
Superclass:
javax.servlet.http.HttpServlet
Browse...
< Back
Next >
Finish
Cancel
Markers
Properties
Servers
Data Source Explorer
Snippets
0 errors, 1 warning, 0 others
Description
Resource
Path
Location
Type
Java Build Path Problems (1 item)
An outline is not available.

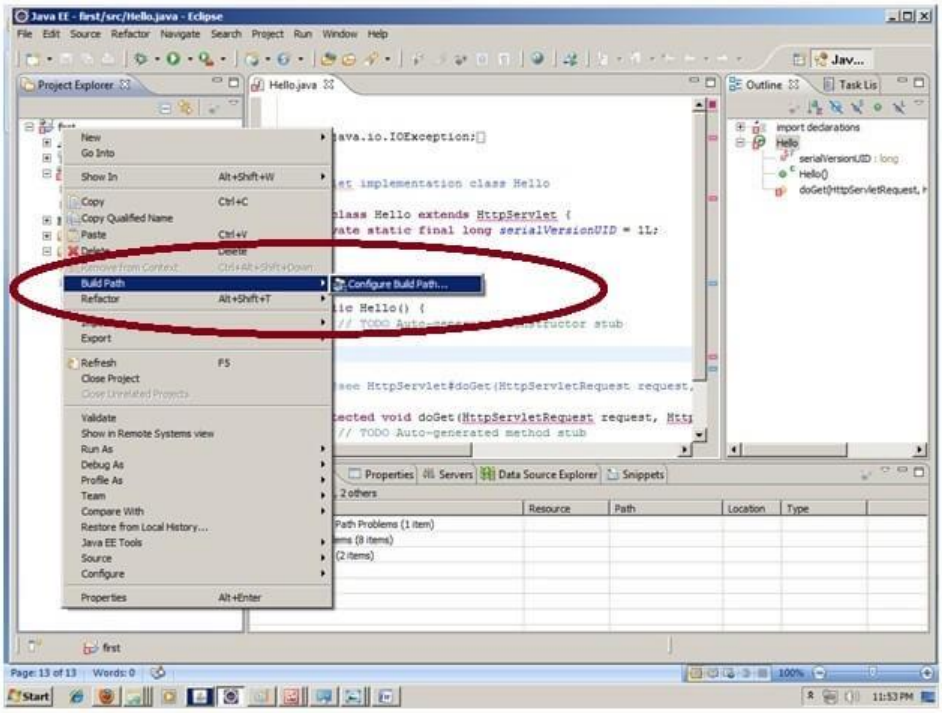

Create Servlet
Create Servlet
Enter servlet deployment descriptor specific information.
Name:
Hello
Description:
Initialization parameters:
Name
Value
Description
Add...
URL mappings:
/Hello
< Back
Next >
Finish
Cancel
An outline is not available.

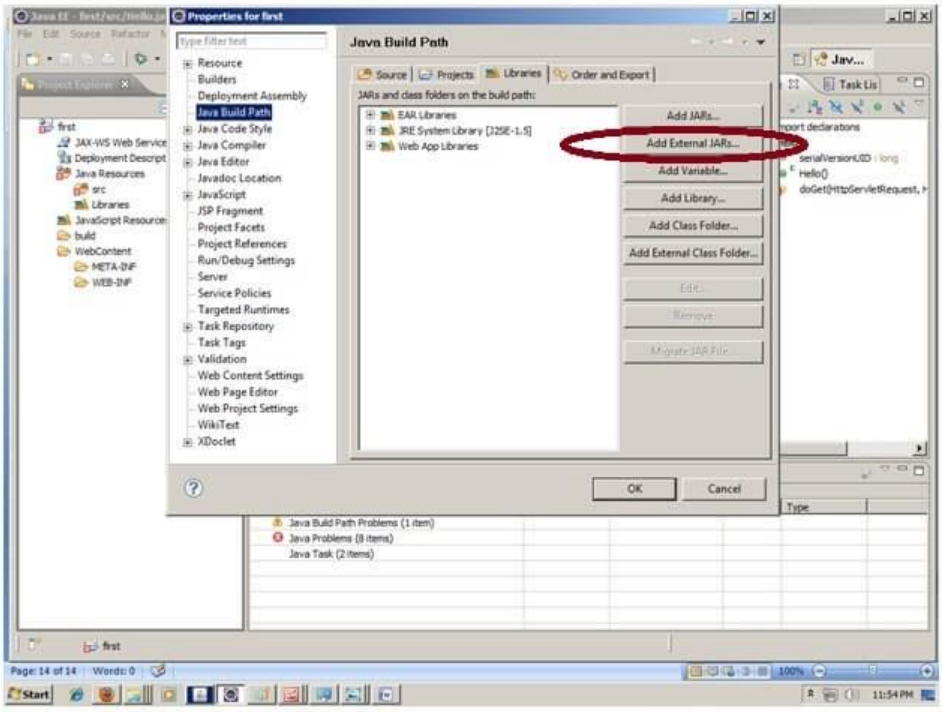


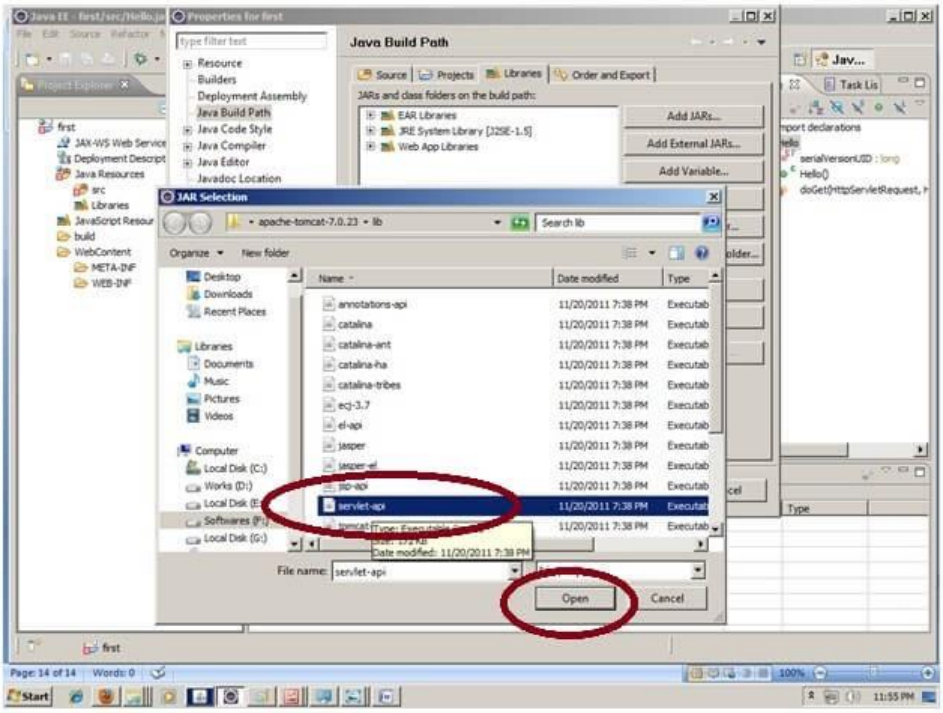


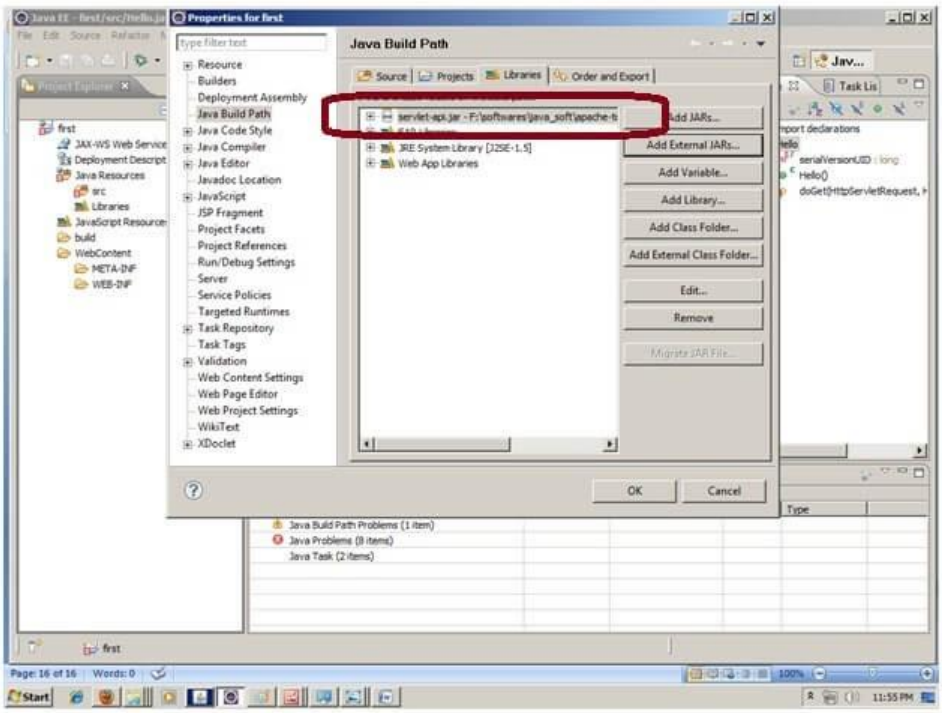
3) add jar file in eclipse IDE:

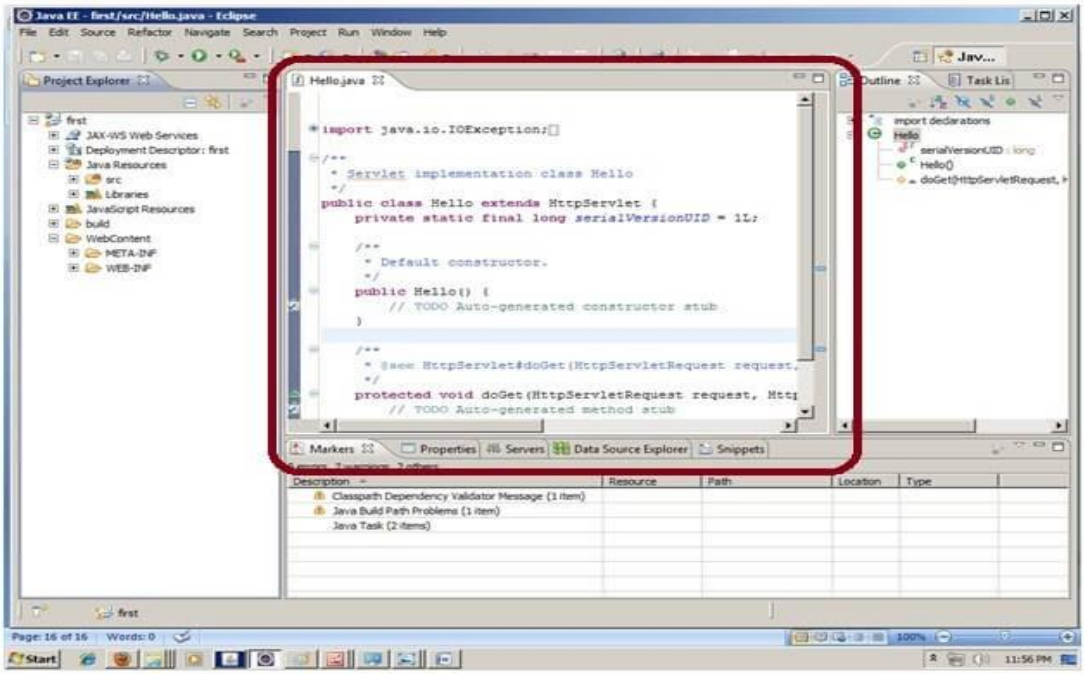
For adding a jar file, **right click on your project -> Build Path -> Configure Build Path > click on Libraries tab in Java Build Path -> click on Add External JARs button -> select the servlet-api.jar file under tomcat/lib -> ok.**


Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
Outline
Task List
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Refactor Alt+Shift+T
Export
Refresh F5
Close Project
Close Unrelated Projects
Validate
Show in Remote Systems view
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter
Configure Build Path...
Properties
Servers
Data Source Explorer
Snippets
Problems
Resource
Path
Location
Type
first
11:53 PM

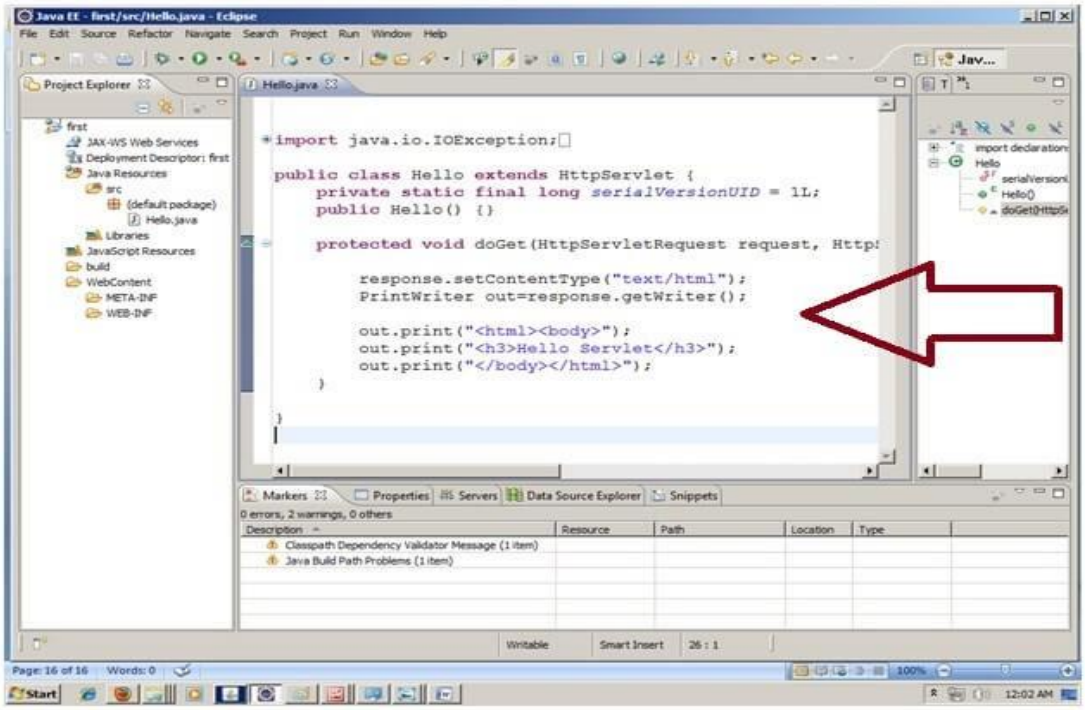

Properties for first
Java Build Path
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Project Facets
Project References
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Page Editor
Web Project Settings
WikiText
XDoclet
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
EAR Libraries
JRE System Library [J2SE-1.5]
Web App Libraries
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel
Java Build Path Problems (1 item)
Java Problems (8 items)
Java Task (2 items)
first
11:54 PM


Java Build Path
JAR Selection
servlet-api
Open
Cancel


Java Build Path
Add External JARs...
OK
Cancel

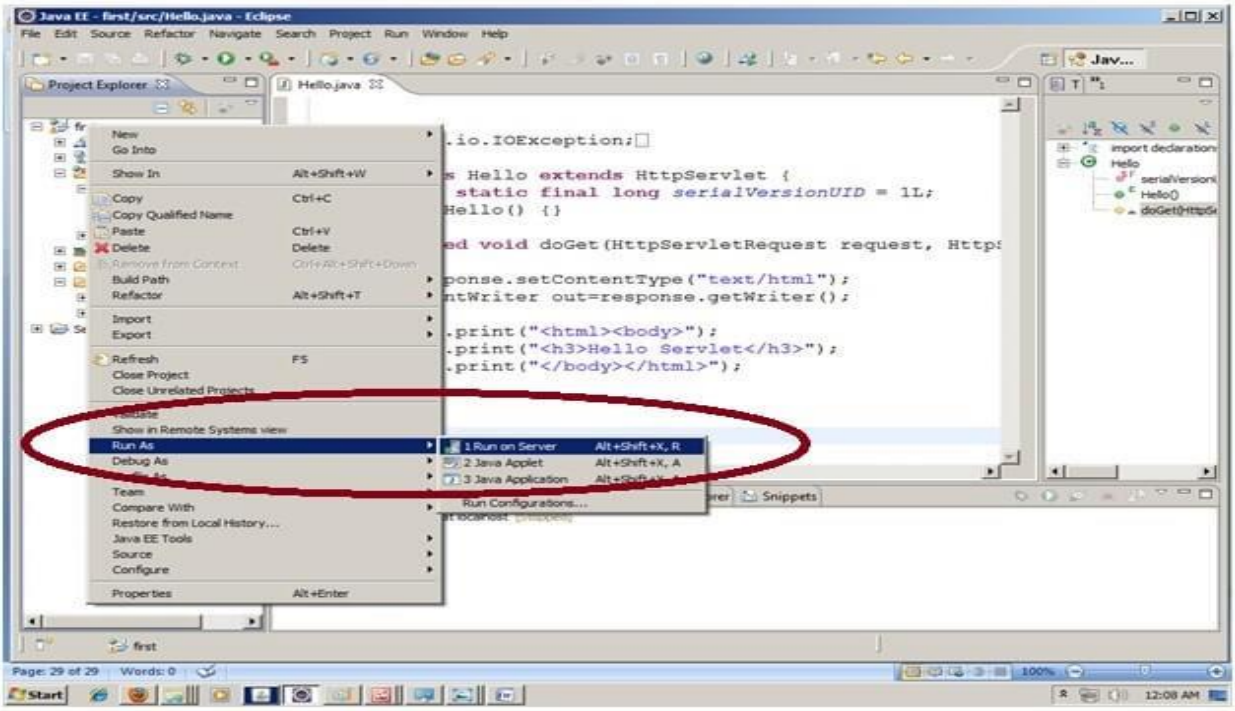


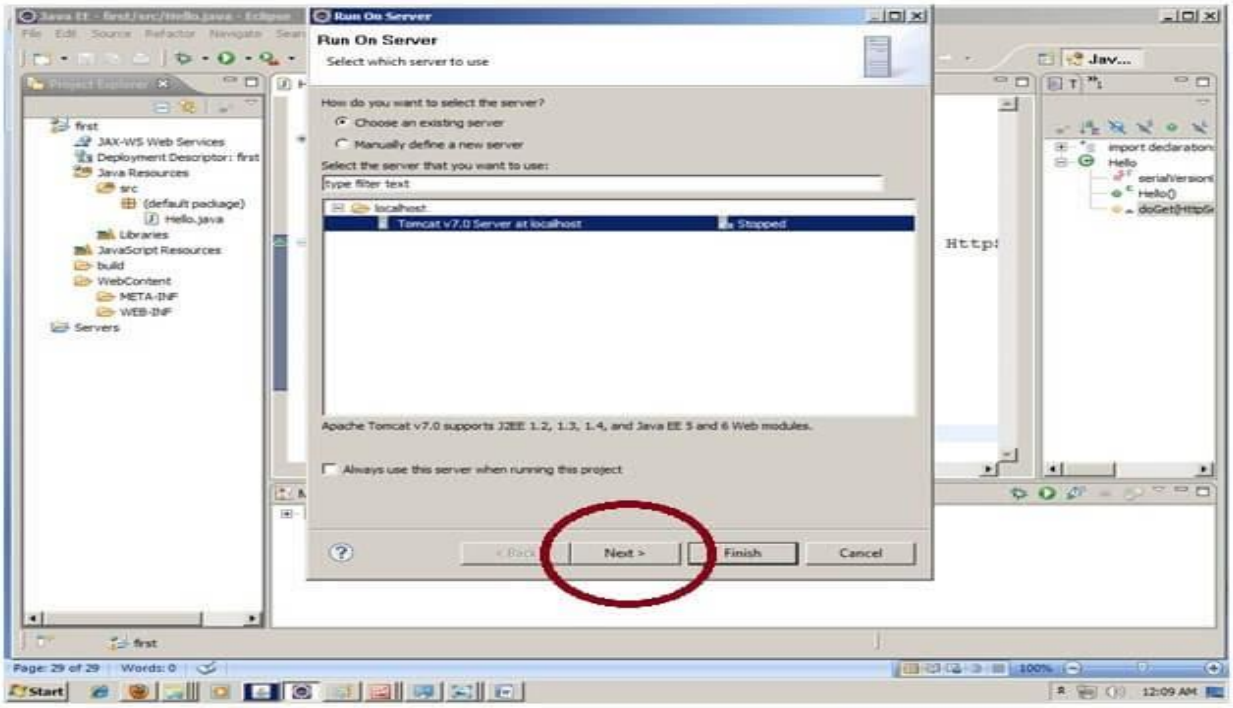
Now servlet has been created, Let's write the first servlet code.

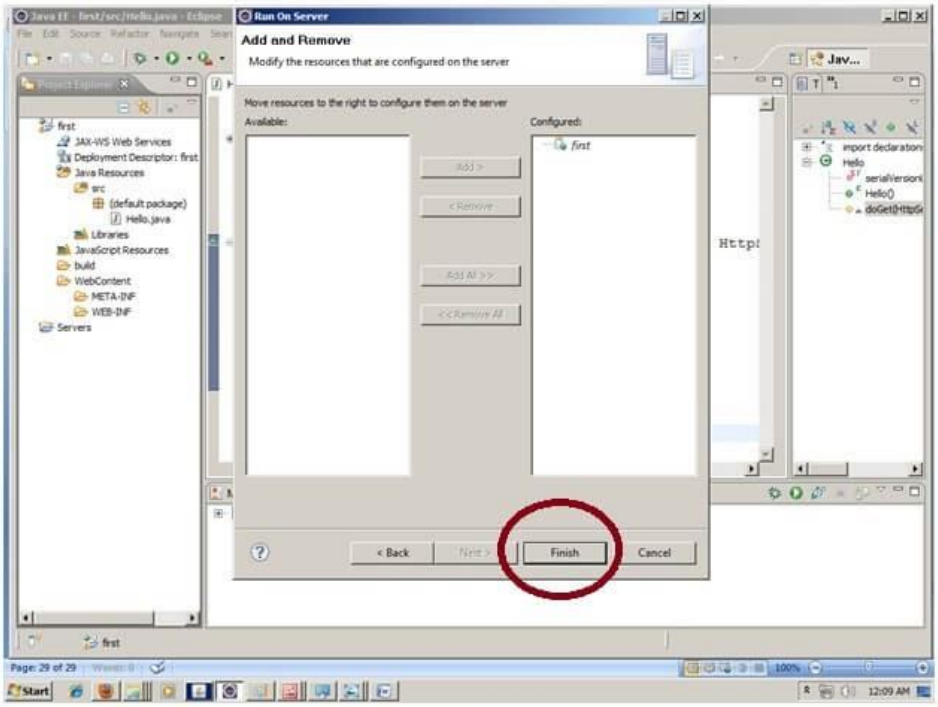


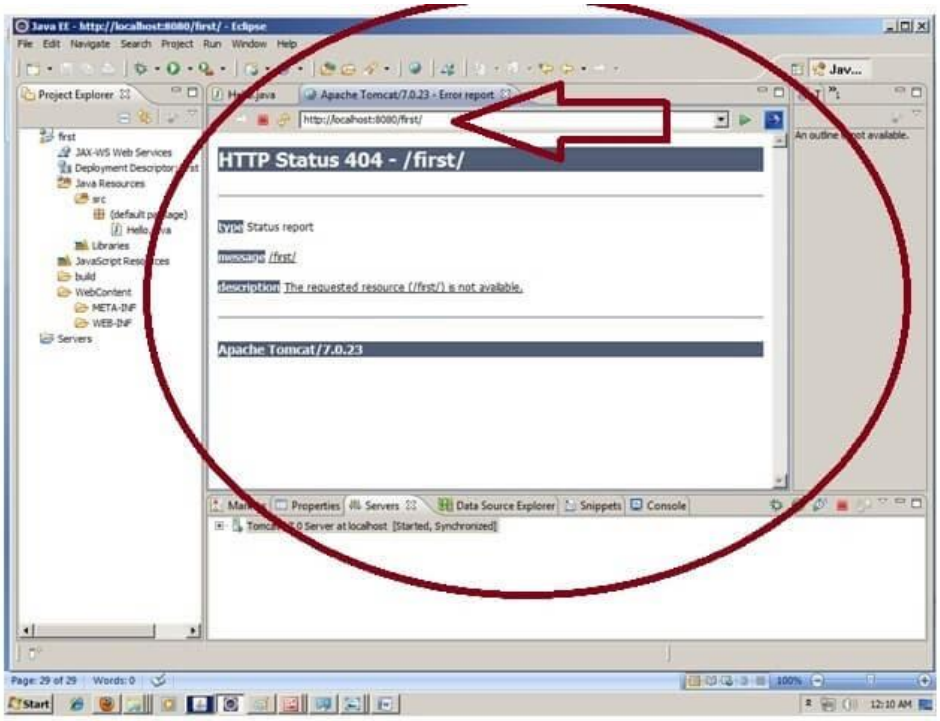
4) Start the server and deploy the project:

For starting the server and deploying the project in one step, **Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.**

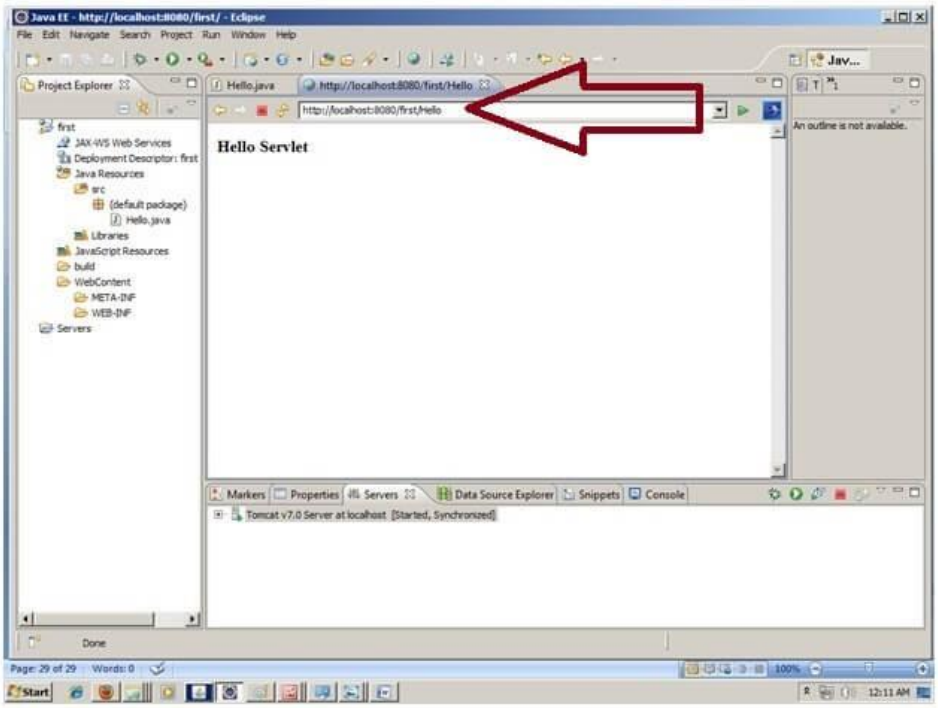

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Refactor Alt+Shift+T
Import
Export
Refresh F5
Close Project
Close Unrelated Projects
Show in Remote Systems view
Run As
Debug As
Team
Compare With
Restore from Local History...
Java EE Tools
Source
Configure
Properties Alt+Enter
1 Run on Server Alt+Shift+X, R
2 Java Applet Alt+Shift+X, A
3 Java Application
Run Configurations...
Snippets
first
Page: 29 of 29 Words: 0
Start
12:08 AM


Run On Server
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
localhost
Tomcat v7.0 Server at localhost
Stopped
Apache Tomcat v7.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5 and 6 Web modules.
Always use this server when running this project
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Servers
Page: 29 of 29 Words: 0
Start
12:09 AM





Now tomcat server has been started and project is deployed. To access the servlet write the url pattern name in the URL bar of the browser. In this case Hello then enter.

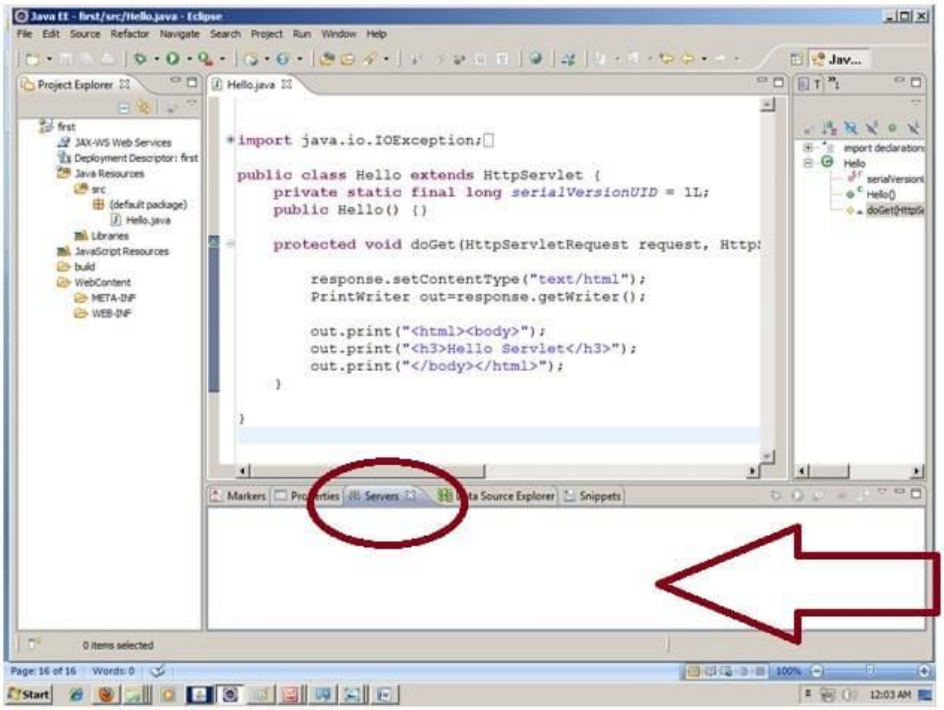


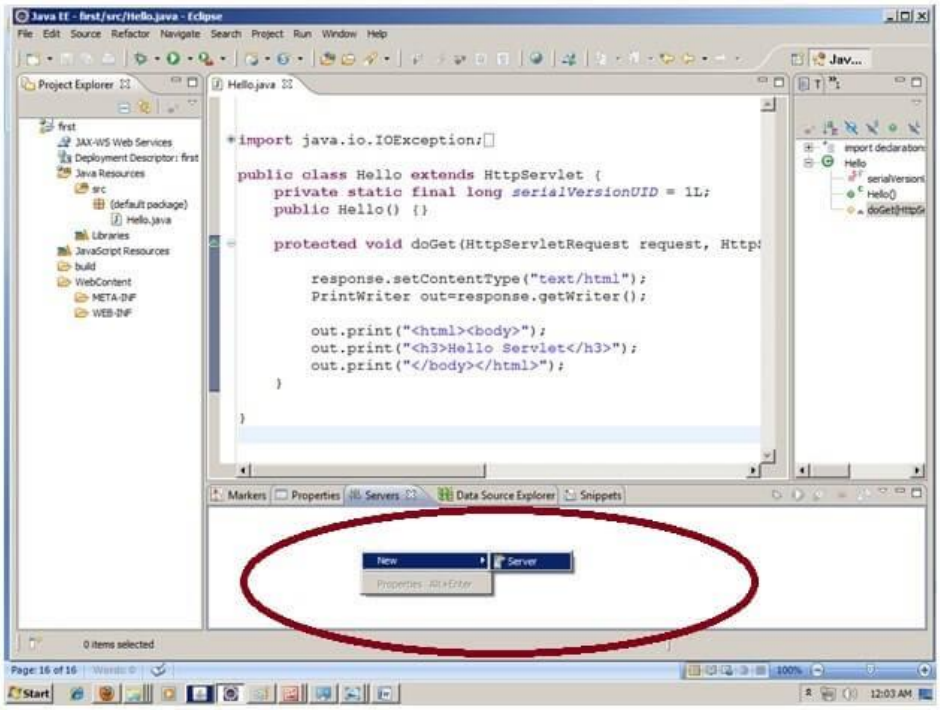
download this example

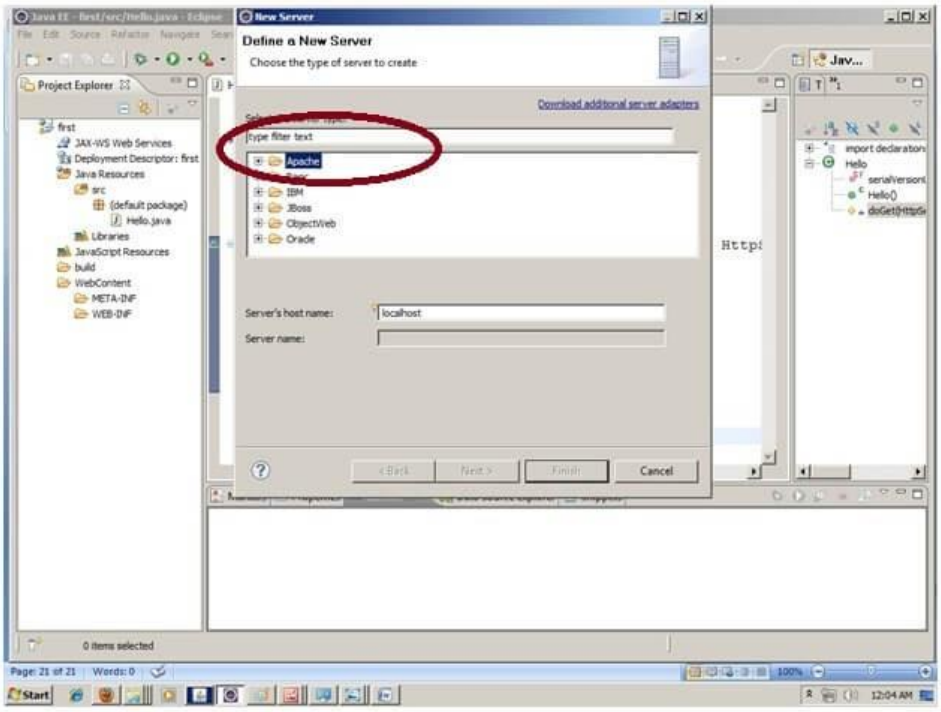
How to configure tomcat server in Eclipse ? (One time Requirement)

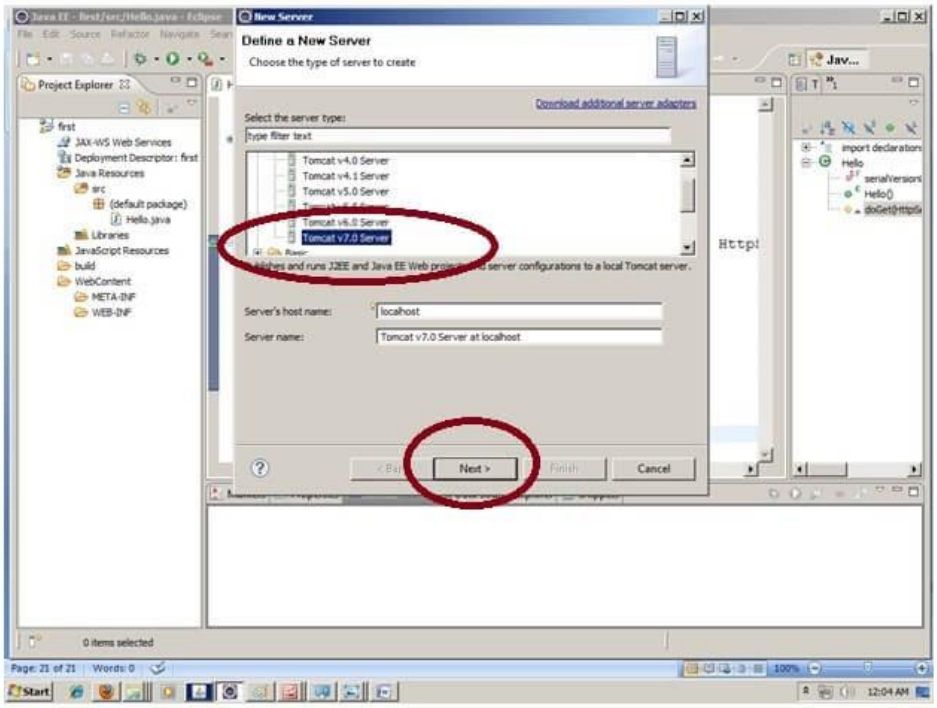
If you are using **Eclipse IDE first time**, you need to configure the tomcat server First.

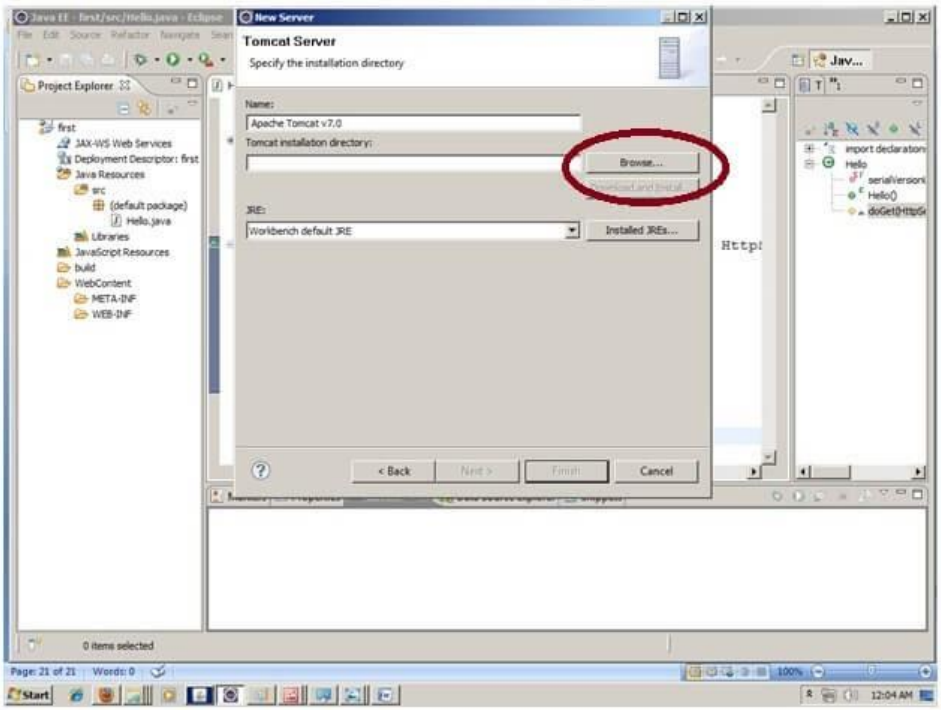
For configuring the tomcat server in eclipse IDE, **click on servers tab at the bottom side of the IDE -> right click on blank area -> New -> Servers -> choose tomcat then its version -> next -> click on Browse button -> select the apache tomcat root folder previous to bin -> next -> addAll -> Finish.**

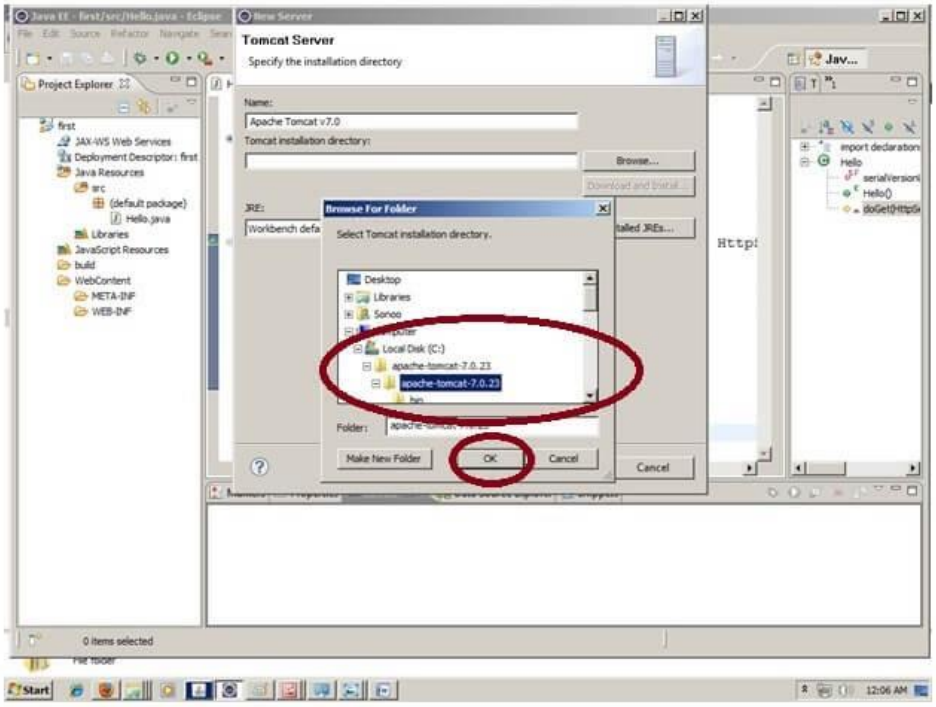

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
import java.io.IOException;
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public Hello() {}
protected void doGet(HttpServletRequest request, Http
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
Markers
Servers
Snippets
0 items selected
Page: 16 of 16
Words: 0
100%
Start
12:03 AM

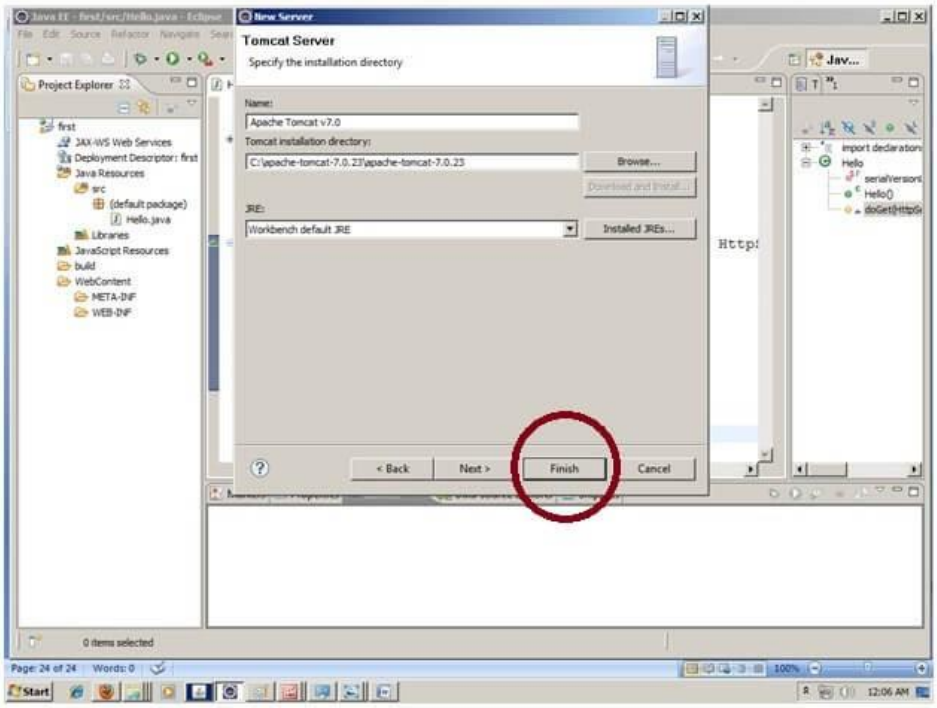

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
import java.io.IOException;
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public Hello() {}
protected void doGet(HttpServletRequest request, Http
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
Markers Properties Servers Data Source Explorer Snippets
New
Server
Properties Alt+Enter
0 items selected
Start
12:03 AM


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
type filter text
Apache
IBM
JBoss
ObjectWeb
Oracle
Server's host name:
localhost
Server name:
< Back
Next >
Finish
Cancel
0 items selected
Start
12:04 AM

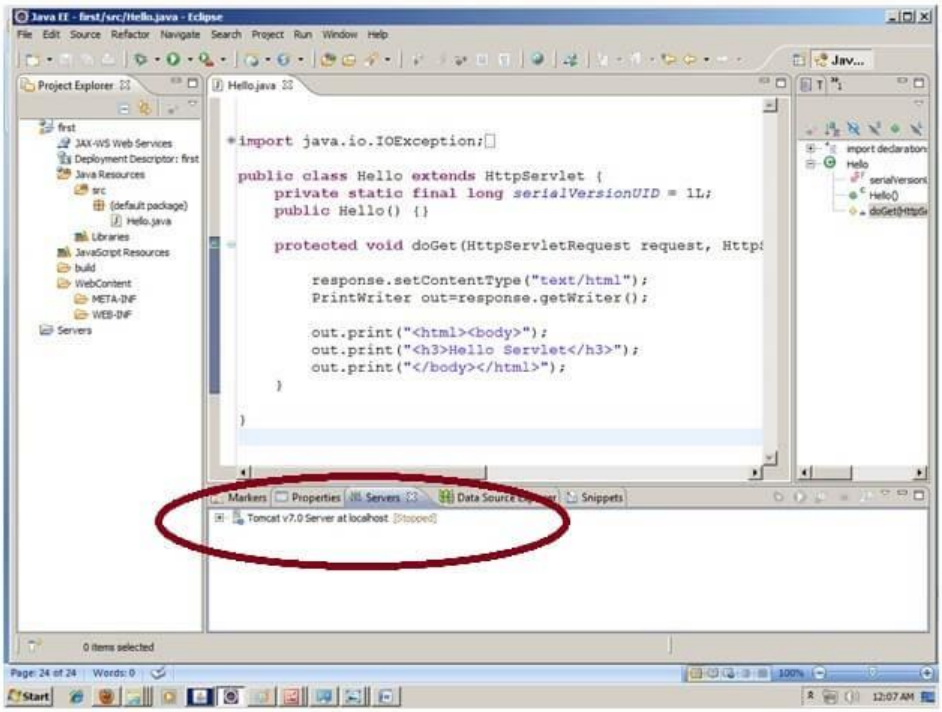

Define a New Server
Choose the type of server to create
Select the server type:
Tomcat v4.0 Server
Tomcat v4.1 Server
Tomcat v5.0 Server
Tomcat v6.0 Server
Tomcat v7.0 Server
Server's host name:
localhost
Server name:
Tomcat v7.0 Server at localhost
Next >
Cancel


Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
Browse...
JRE:
Workbench default JRE
Installed JREs...
< Back
Next >
Finish
Cancel





Now tomcat7 server has been configured in eclipse IDE.


Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Servers
import java.io.IOException;
public class Hello extends HttpServlet {
private static final long serialVersionUID = 1L;
public Hello() {}
protected void doGet(HttpServletRequest request, Http
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
Markers
Properties
Servers
Data Source
Snippets
Tomcat v7.0 Server at localhost [Stopped]
0 items selected
Page: 24 of 24
Words: 0
100%
Start
12:07 AM

Creating Servlet in myeclipse IDE

You need to follow the following steps to create the servlet in the myeclipse IDE. The steps are as follows:

- Create a web
- project create a
- html file create a

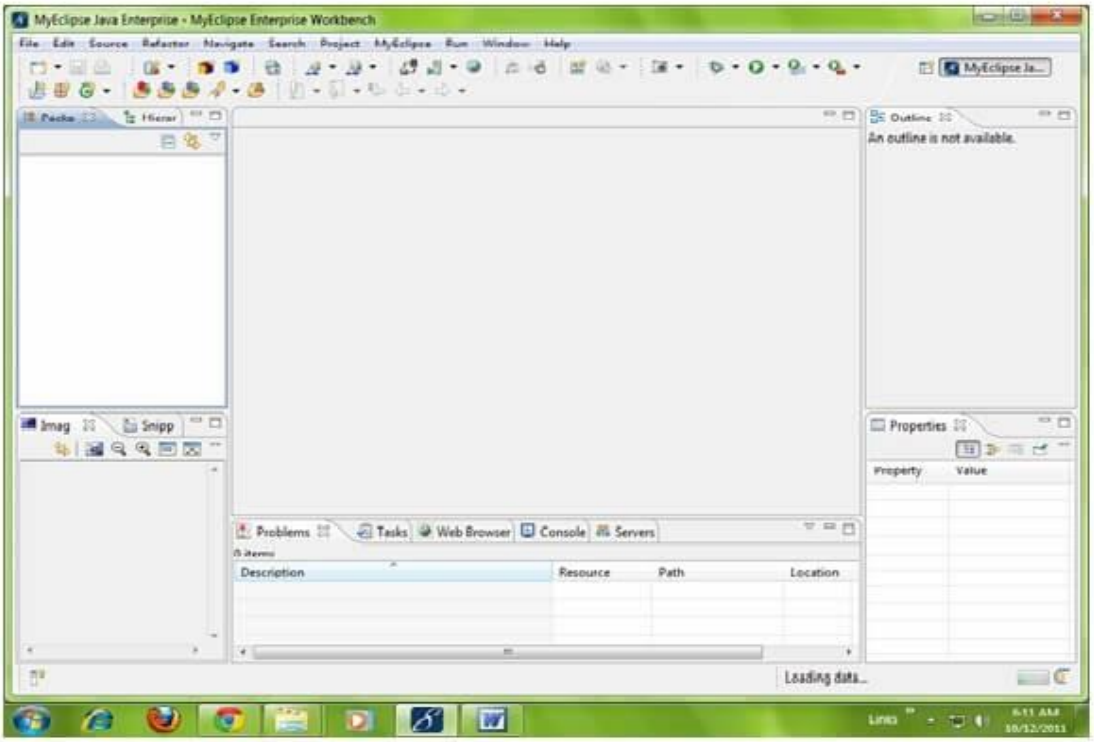
servlet

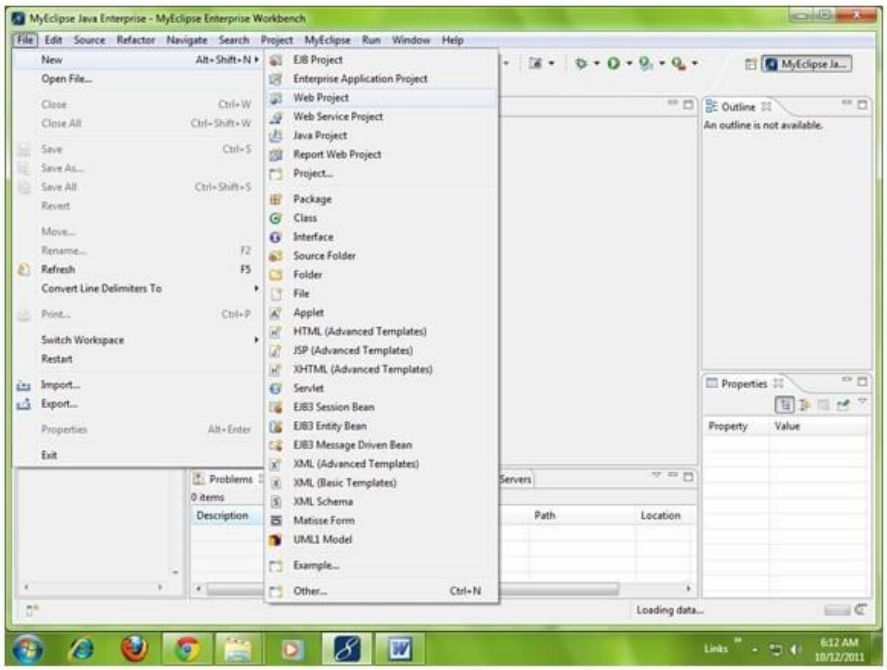
start myeclipse tomcat server and deploy project

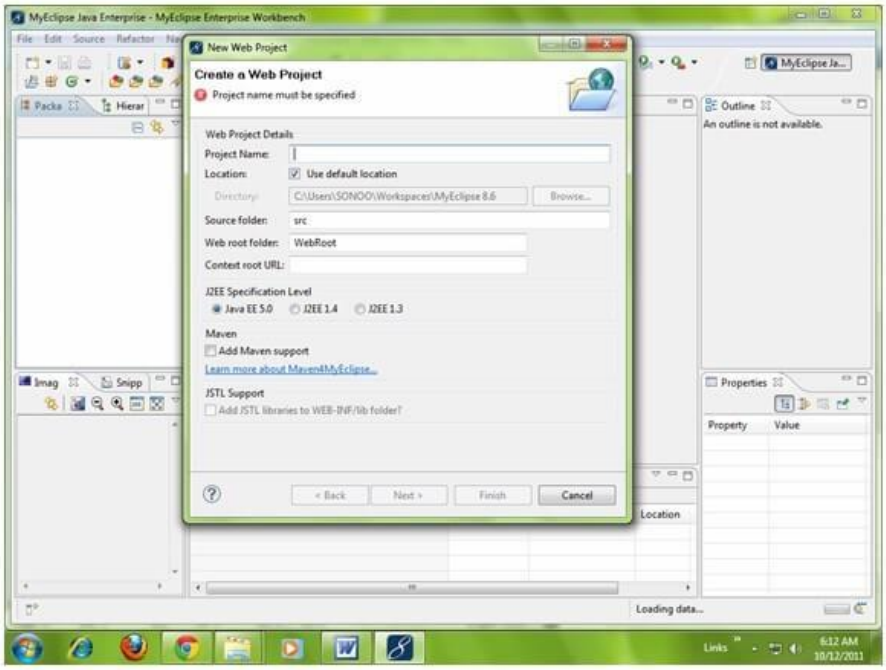
download this example

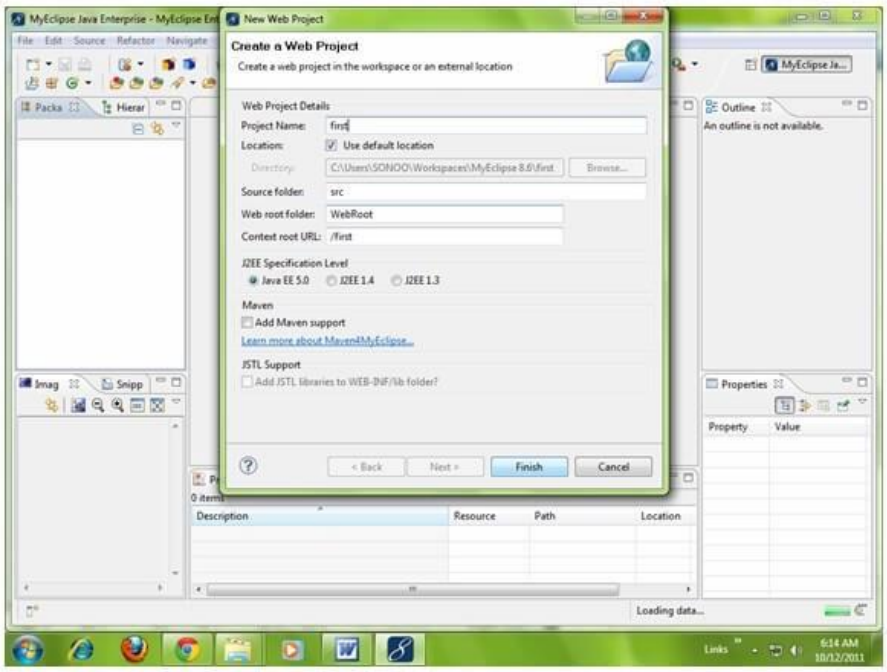
1) Create the web project:

For creating a web project click on File Menu -> New -> web project -> write your project name e.g. first -> Finish.



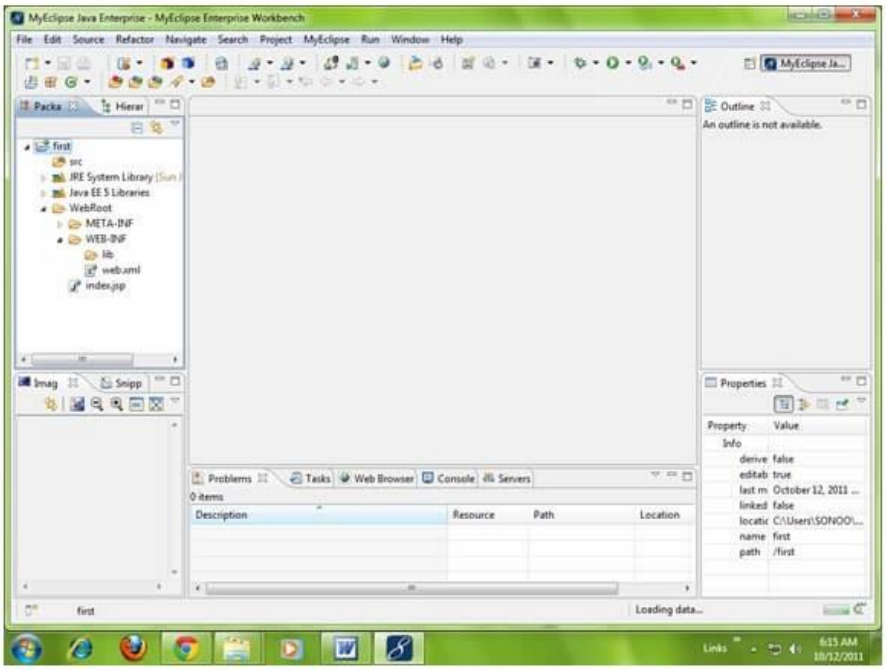

MyEclipse Java Enterprise - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Switch Workspace
Restart
Import...
Export...
Properties Alt+Enter
Exit
EJB Project
Enterprise Application Project
Web Project
Web Service Project
Java Project
Report Web Project
Project...
Package
Class
Interface
Source Folder
Folder
File
Applet
HTML (Advanced Templates)
JSP (Advanced Templates)
XHTML (Advanced Templates)
Servlet
EJB3 Session Bean
EJB3 Entity Bean
EJB3 Message Driven Bean
XML (Advanced Templates)
XML (Basic Templates)
XML Schema
Matisse Form
UML1 Model
Example...
Other... Ctrl+N
Outline
An outline is not available.
Properties
Property Value
Problems
0 items
Description
Path
Location
Servers
Loading data...
Links
6:12 AM
10/12/2011


MyEclipse Java Enterprise - MyEclipse Enterprise Workbench
New Web Project
Create a Web Project
Project name must be specified
Web Project Details
Project Name:
Location:
Use default location
Directory:
C:\Users\SONOO\Workspaces\MyEclipse 8.6
Browse...
Source folder:
src
Web root folder:
WebRoot
Context root URL:
J2EE Specification Level
Java EE 5.0
J2EE 1.4
J2EE 1.3
Maven
Add Maven support
Learn more about Maven4MyEclipse...
JSTL Support
Add JSTL libraries to WEB-INF/lib folder?
< Back
Next >
Finish
Cancel
Outline
An outline is not available.
Properties
Property Value
Location
Loading data...
Links
6:12 AM
10/12/2011

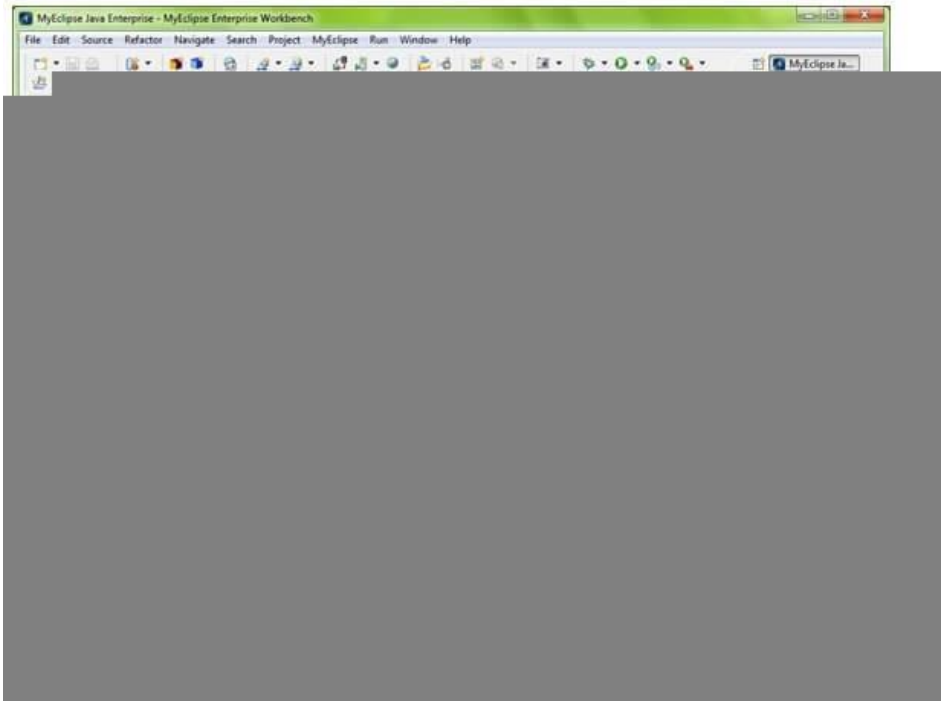


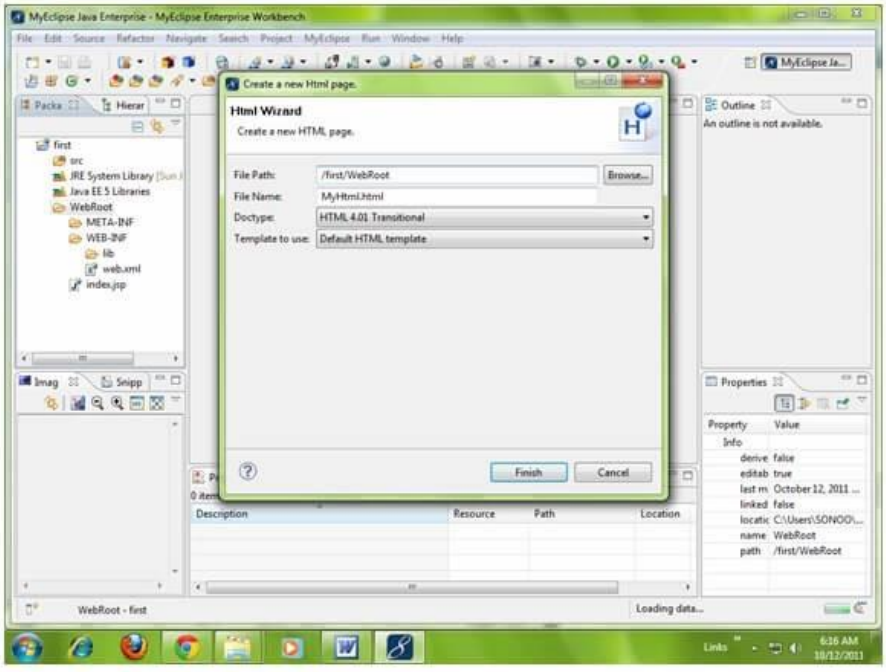
2) Create the html file:

As you can see that a project is created named first. Now let's explore this project.

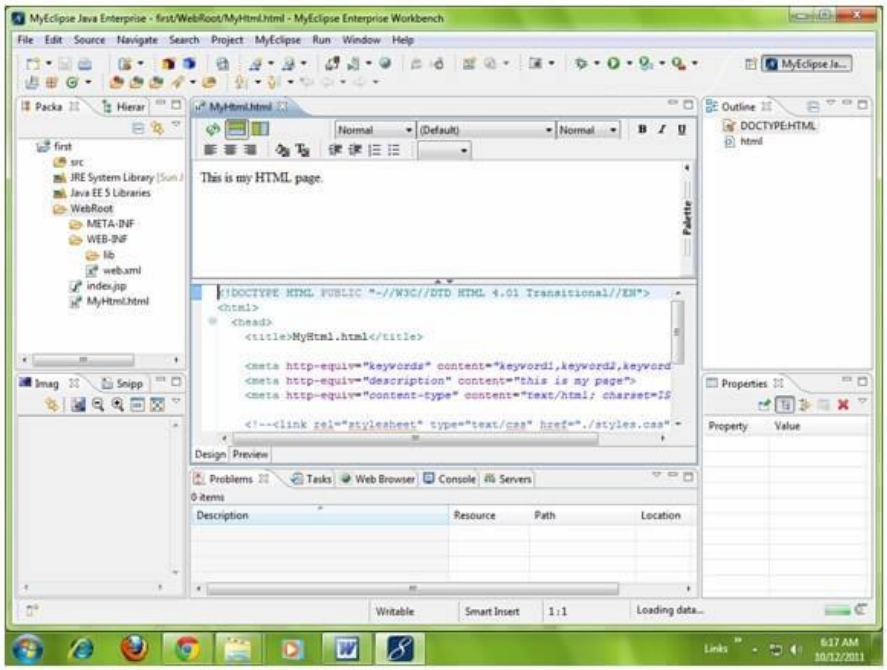


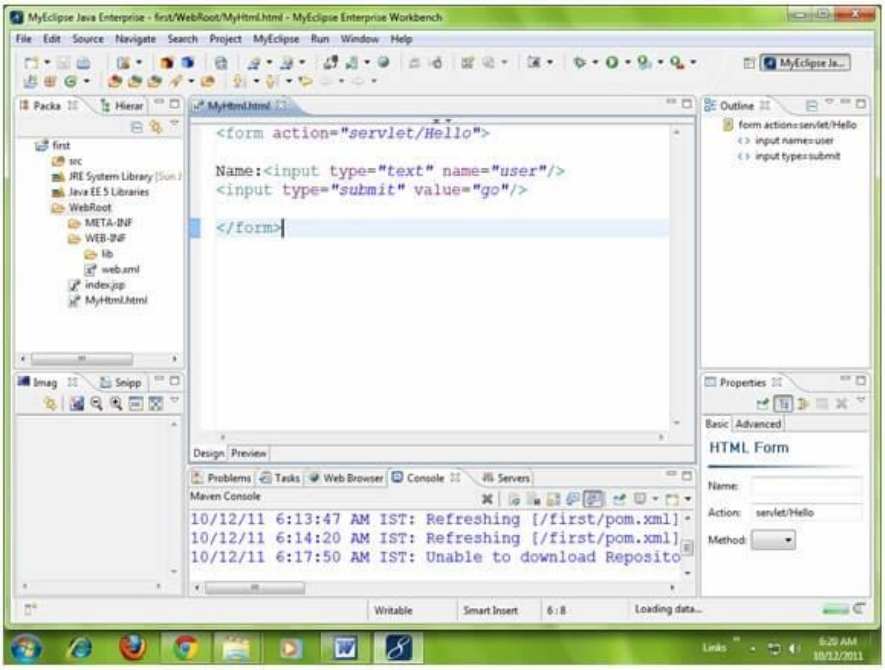
For creating a html file, right click on WebRoot -> New -> html -> write your html file name e.g. MyHtml.html -> Finish.





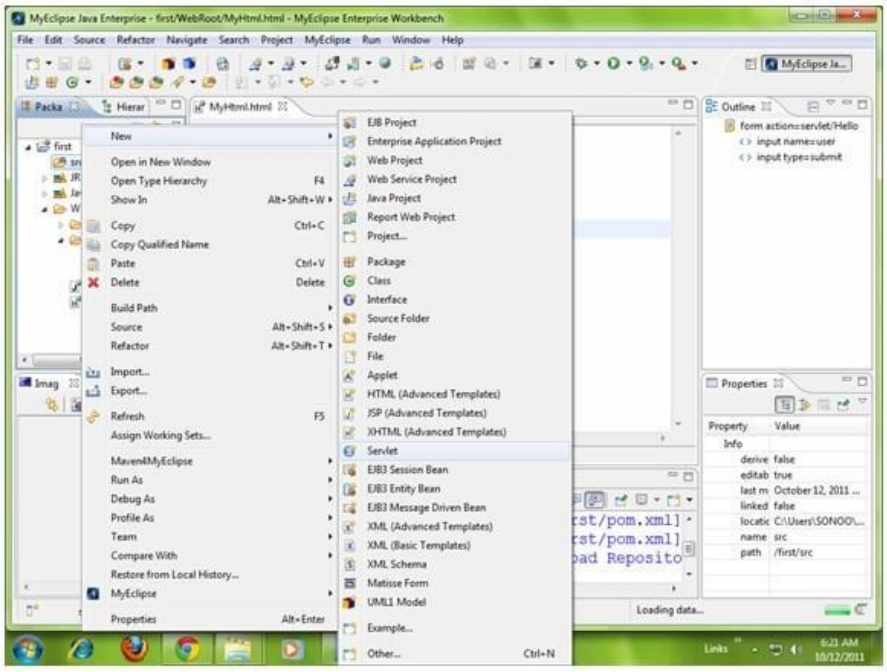
As you can see that a html file is created named MyHtml.html. Now let's write the html code here.

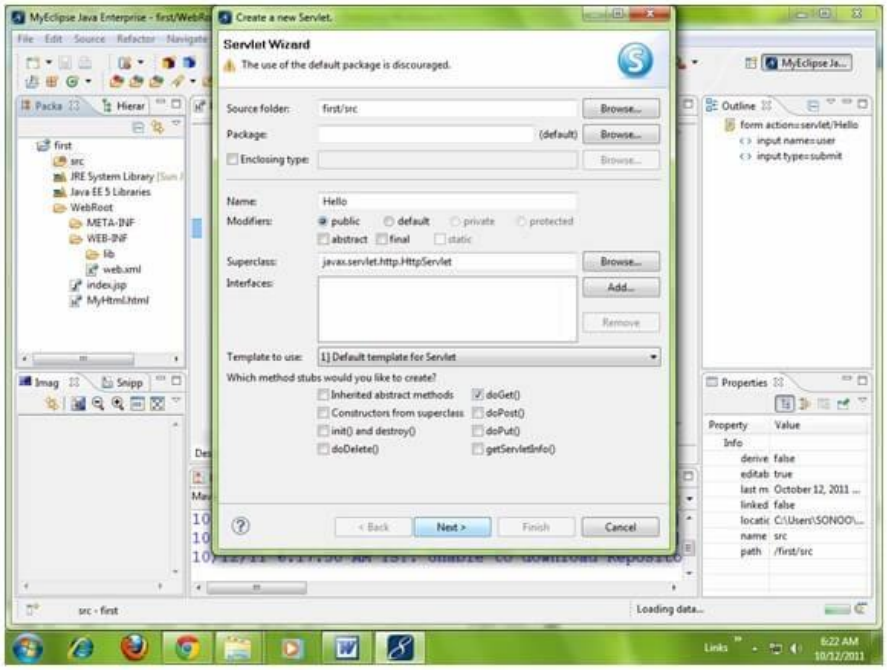


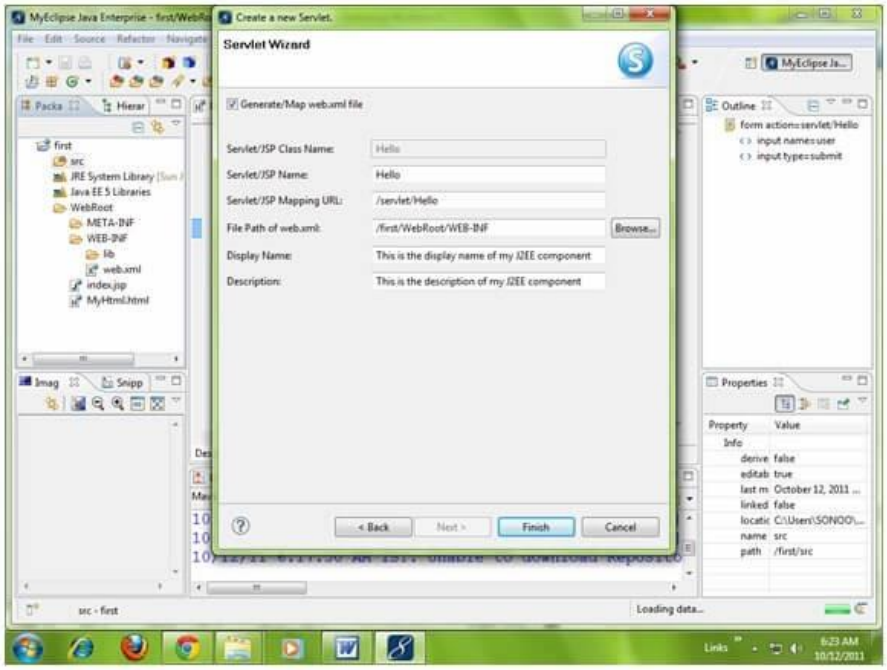


3) Create the servlet:

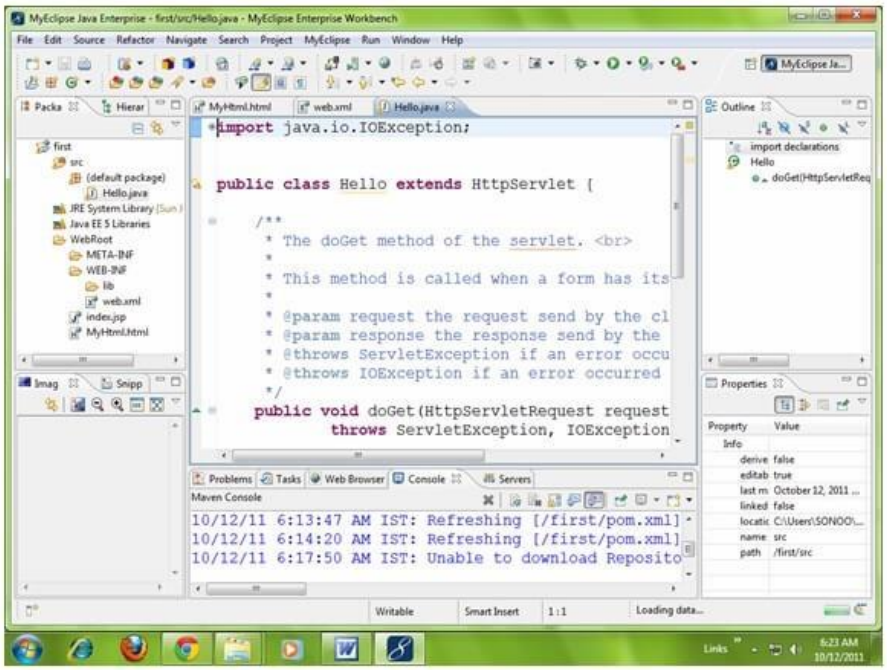
For creating a servlet click on File Menu -> New -> servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish.

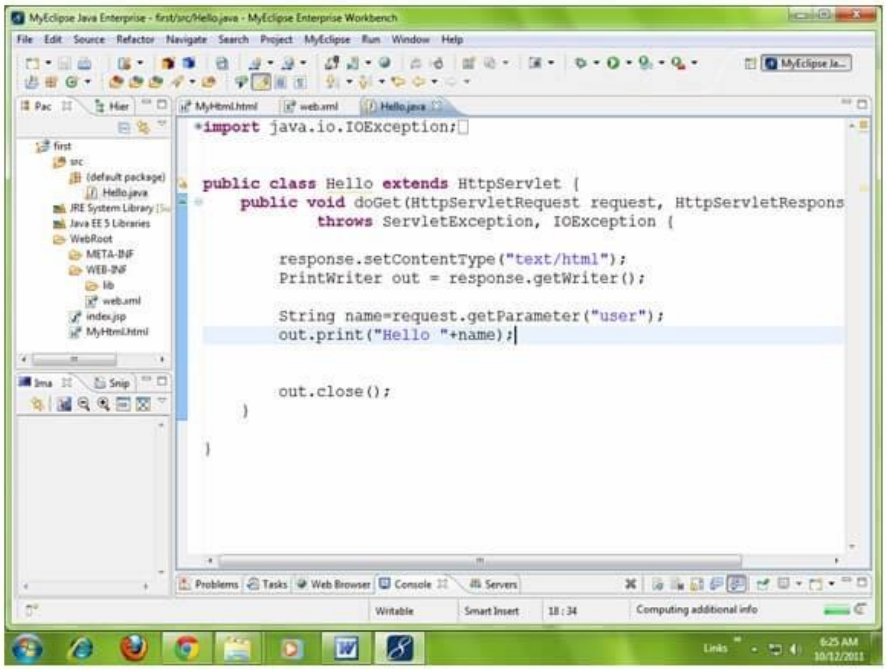

MyEclipse Java Enterprise - first/WebRoot/MyHtml.html - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Maven4MyEclipse
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
MyEclipse
Properties Alt+Enter
EJB Project
Enterprise Application Project
Web Project
Web Service Project
Java Project
Report Web Project
Project...
Package
Class
Interface
Source Folder
Folder
File
Applet
HTML (Advanced Templates)
JSP (Advanced Templates)
XHTML (Advanced Templates)
Servlet
EJB3 Session Bean
EJB3 Entity Bean
EJB3 Message Driven Bean
XML (Advanced Templates)
XML (Basic Templates)
XML Schema
Matisse Form
UML1 Model
Example...
Other... Ctrl+N
Outline
form action=servlet/Hello
input name=user
input type=submit
Properties
Property Value
Loading data...


Create a new Servlet.
Servlet Wizard
The use of the default package is discouraged.
Source folder: first/src
Package: (default)
Enclosing type:
Name: Hello
Modifiers: public default private protected
abstract final static
Superclass: javax.servlet.http.HttpServlet
Interfaces:
Template to use: 1) Default template for Servlet
Which method stubs would you like to create?
Inherited abstract methods
Constructors from superclass
init() and destroy()
doDelete()
doGet()
doPost()
doPut()
getServletInfo()
< Back
Next >
Finish
Cancel
Outline
Properties
Loading data...



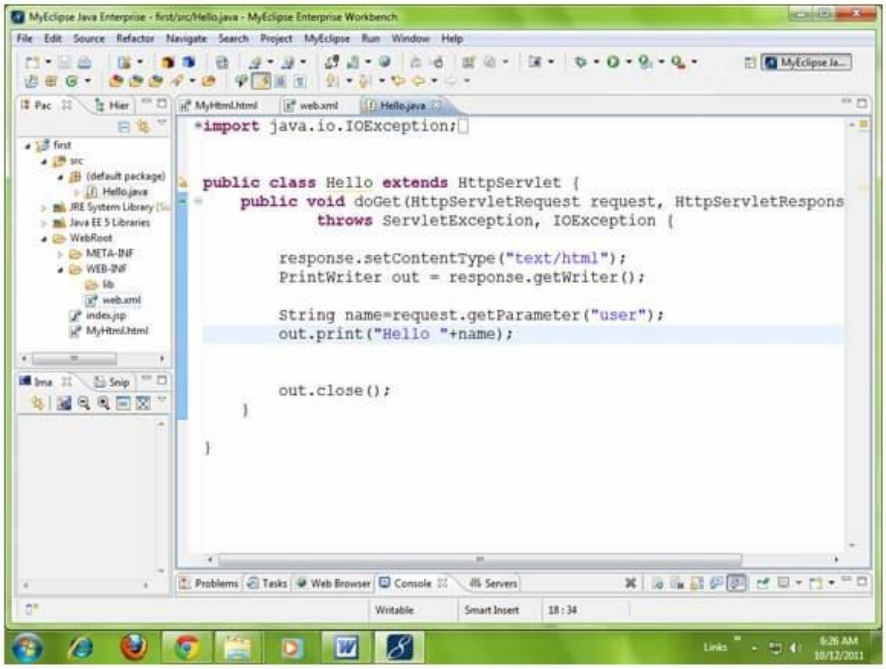
As you can see that a servlet file is created named Hello.java. Now let's write the servlet code here.



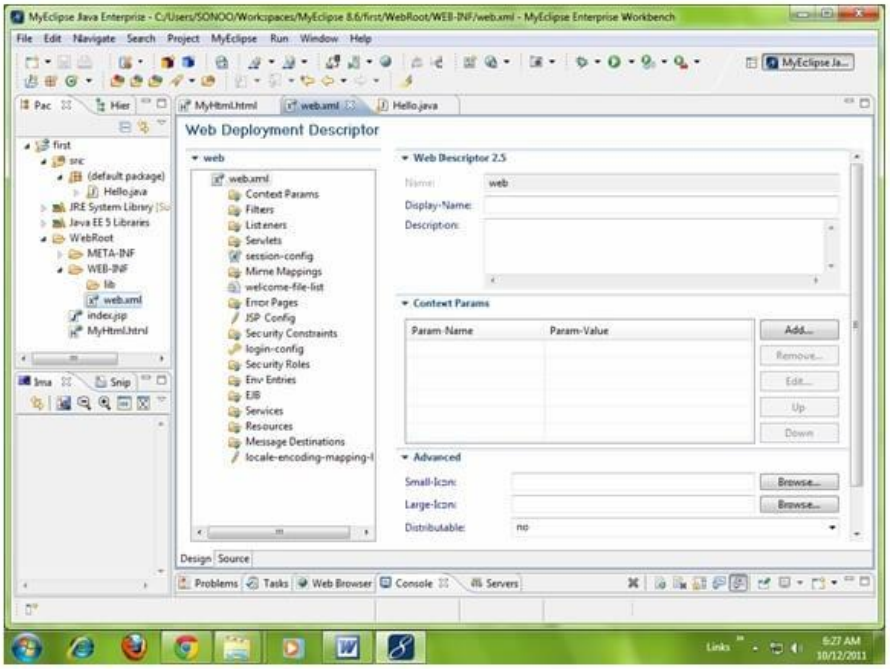


Now let's make the MyHtml.html file as the default page of our project.

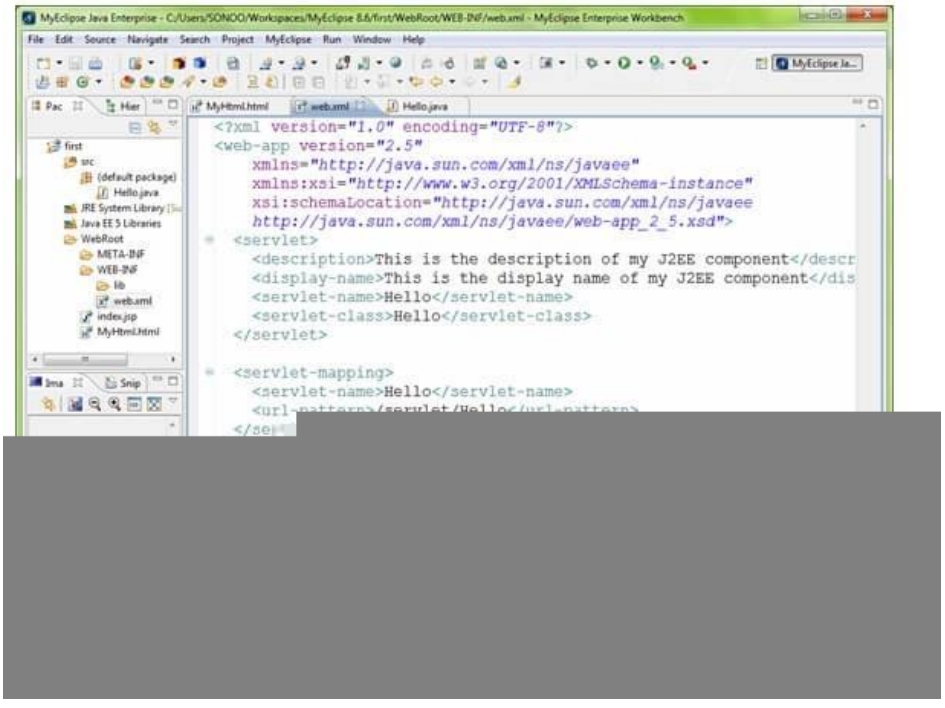
For this, open web.xml file and change the welcome file name as MyHtml.html in place of index.jsp.

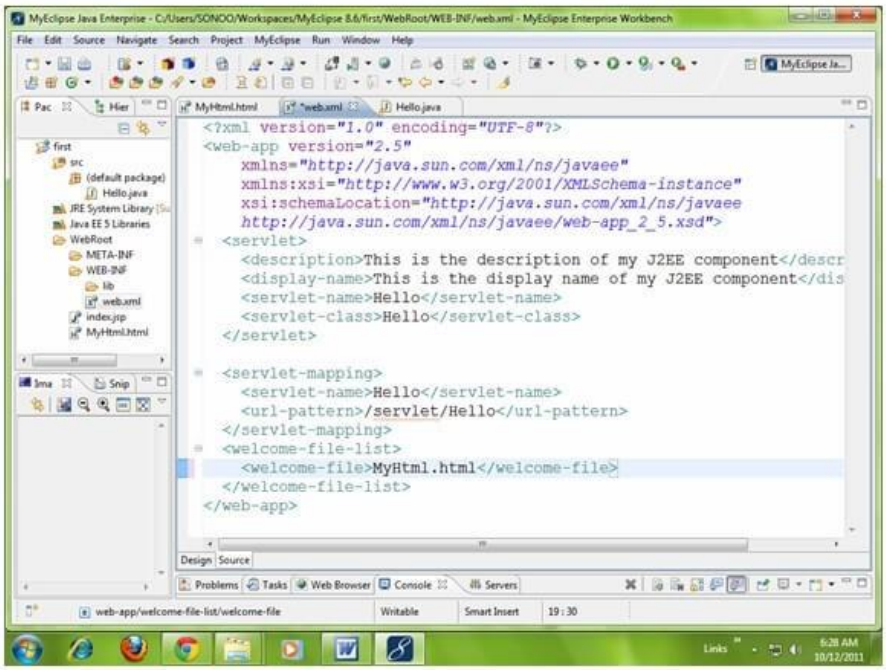


Click on the source tab to see the source code.



Now change the welcome file as MyHtml.html in place of index.jsp.

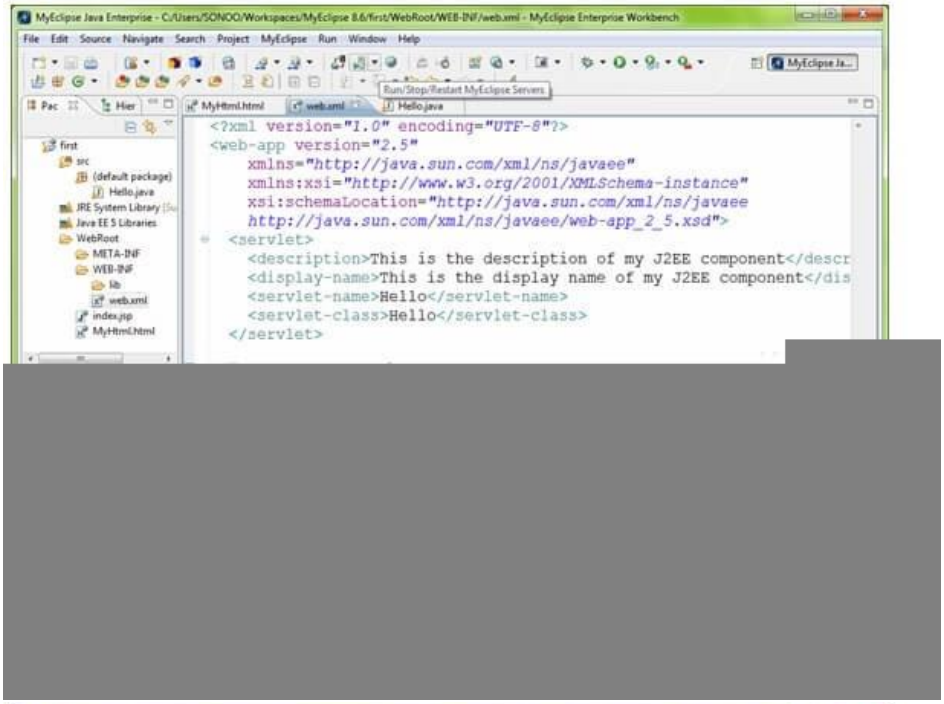


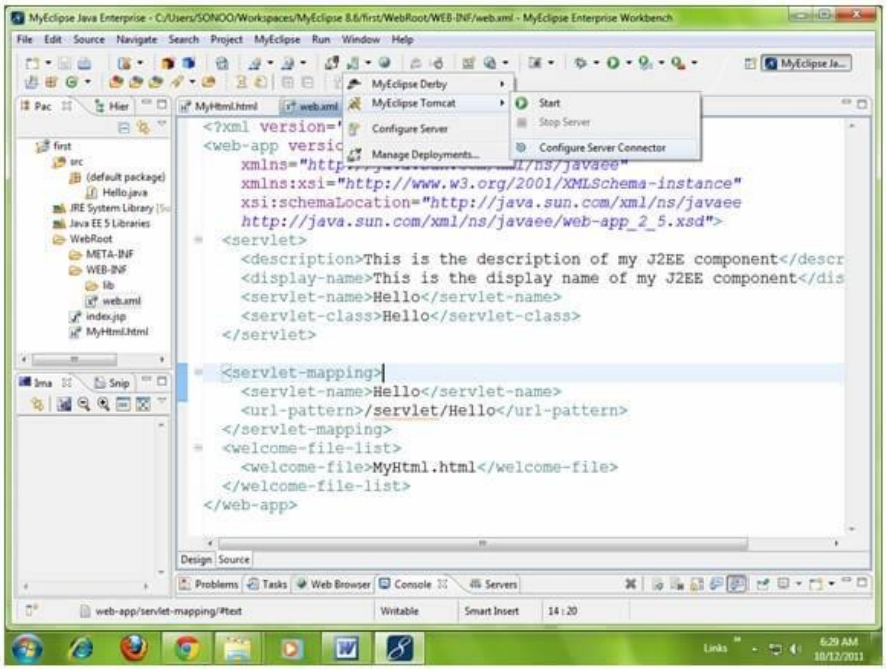


4) Start the server and deploy the project:

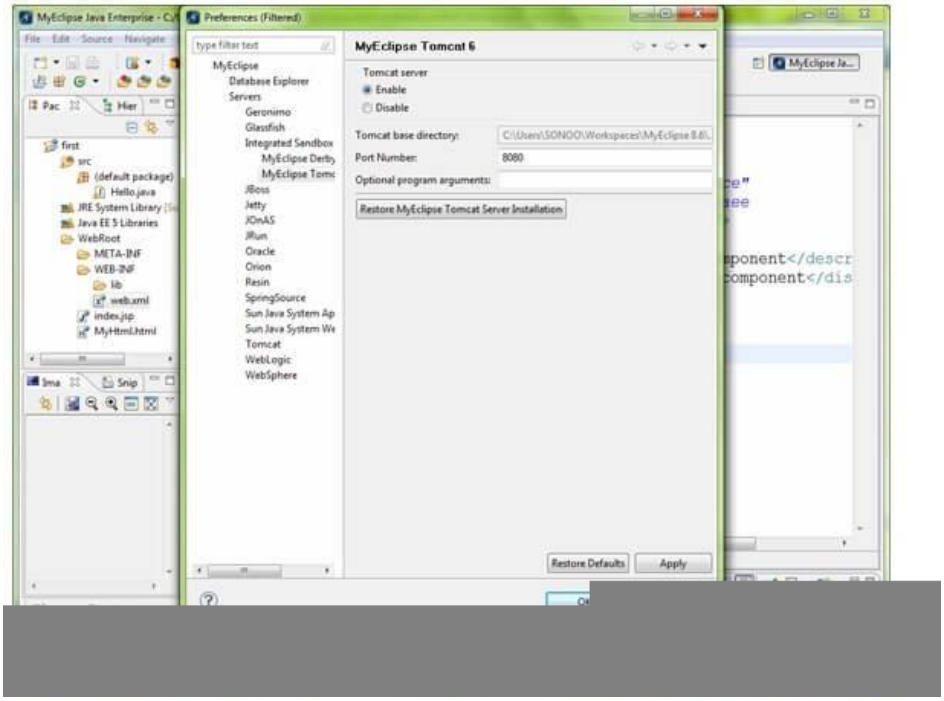
For starting the server and deploying the project in one step Right click on your project -> Run As -> MyEclipse server application.

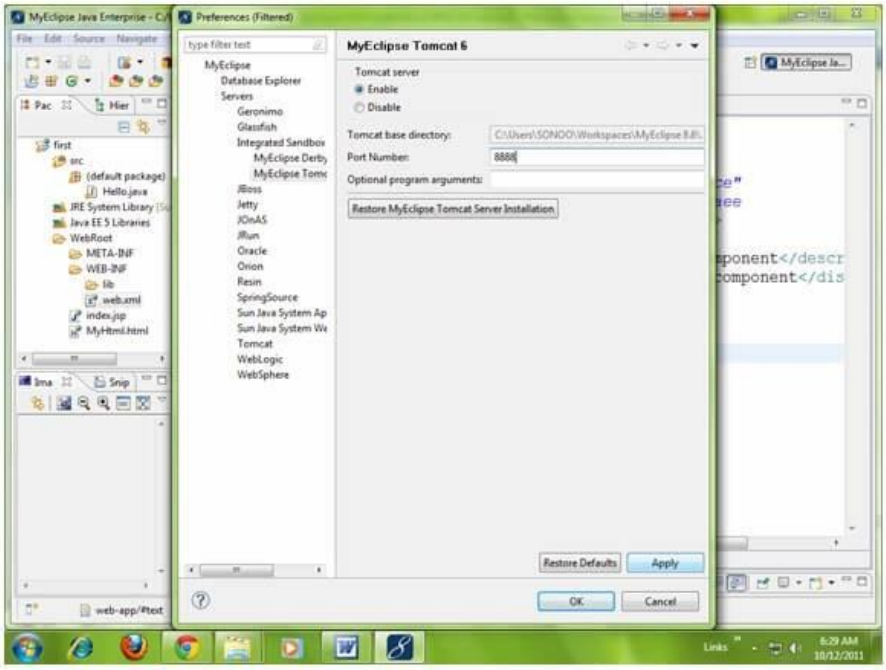
The default port of myeclipse tomcat is 8080, if you have installed oracle on your system, the port no. will conflict so let's first change the port number of myeclipse tomcat server. For changing the port number click on the start server icon at the left hand side of browser icon > myeclipse tomcat -> Configure server connector -> change the port number as 8888 in place of 8080 -> apply -> ok.



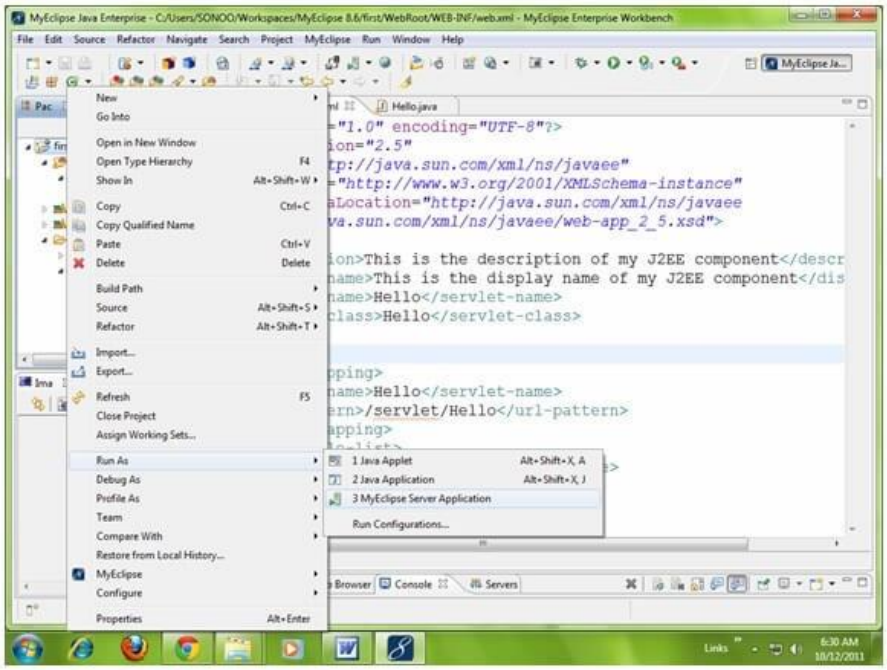


Now change the port number as 8888 in place of 8080 -> apply -> ok.

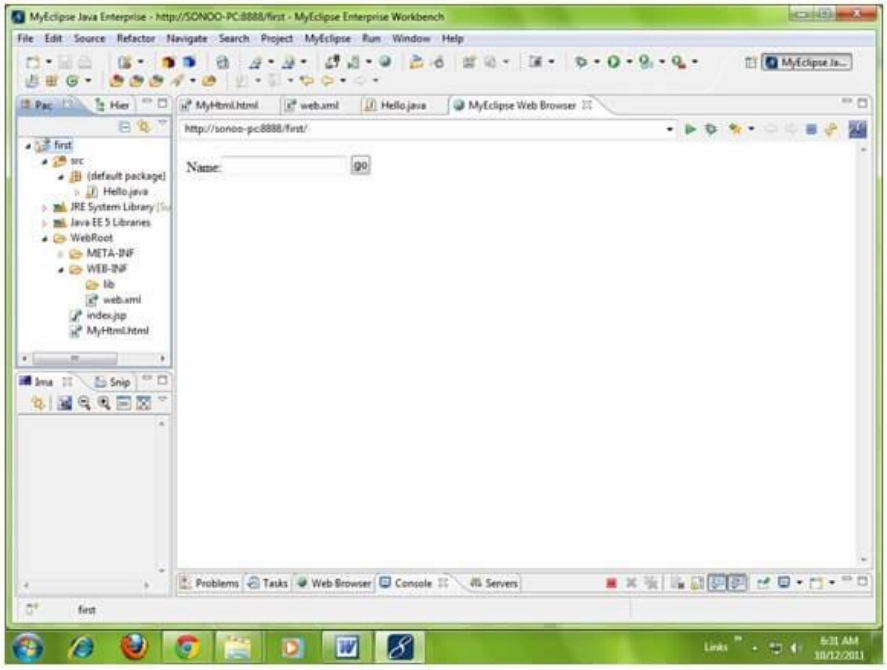


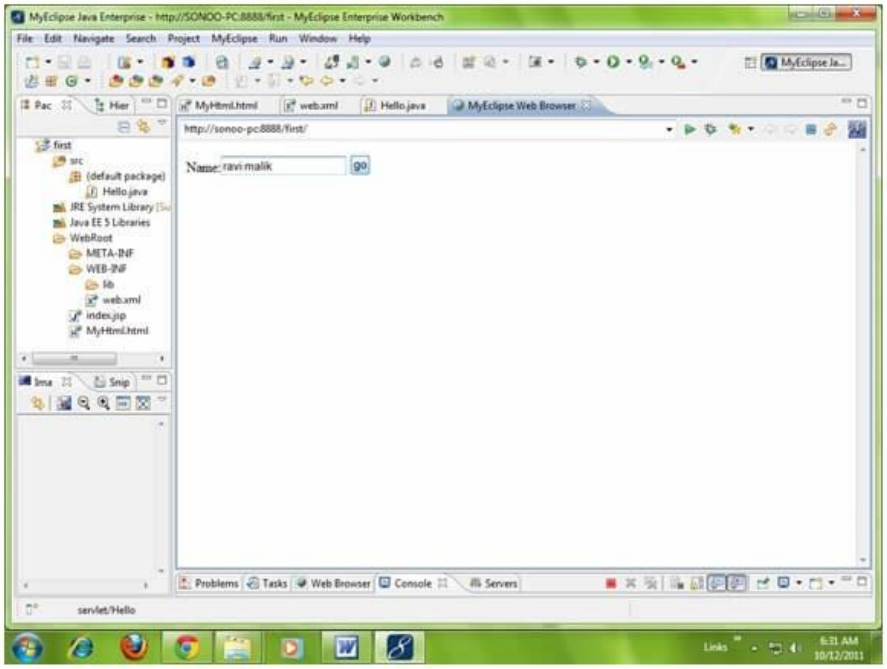


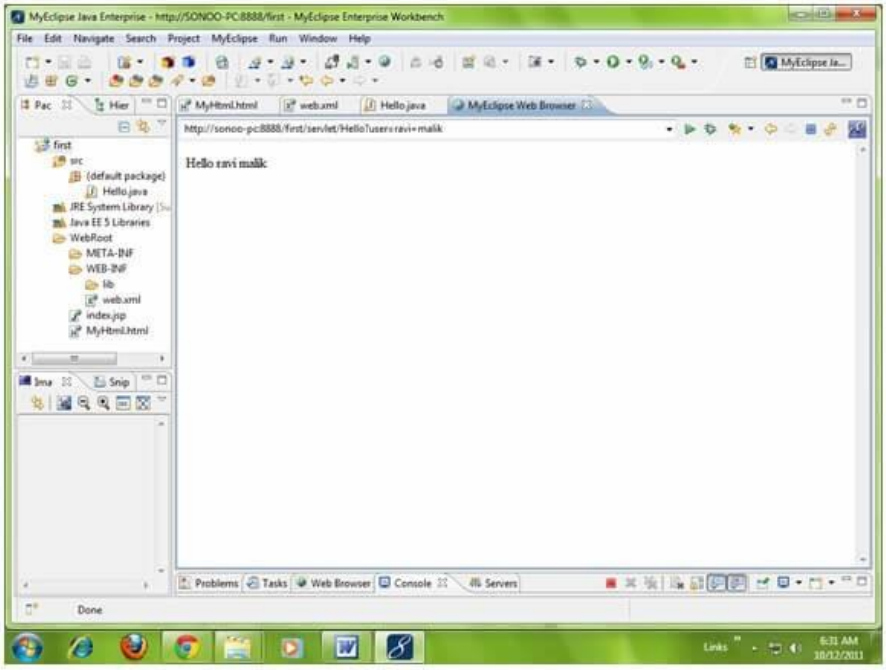
Now port number have been changed. For starting the server Right click on your project -> Run As -> MyEclipse server application.



As you can see that default page of your project is open, write your name -> go.


MyEclipse Java Enterprise - http://SONOO-PC:8888/first - MyEclipse Enterprise Workbench
Name:
go
first


MyEclipse Java Enterprise - http://SONOO-PC:8888/first - MyEclipse Enterprise Workbench
Name: ravi malik
go
servlet/Hello



download this example

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

Method	Description

public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Example of ServletRequest to display the name of the user

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

index.html

```
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
```

DemoServ.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServ extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter pw=res.getWriter();

String name=req.getParameter("name");//will return value
pw.println("Welcome "+name);

pw.close();
}}
```

RequestDispatcher in Servlet

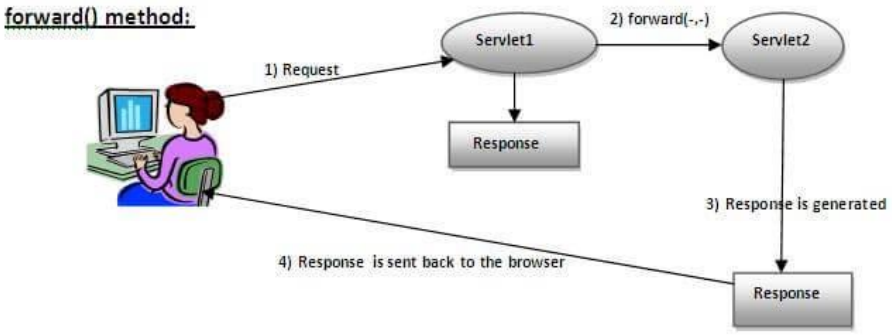
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

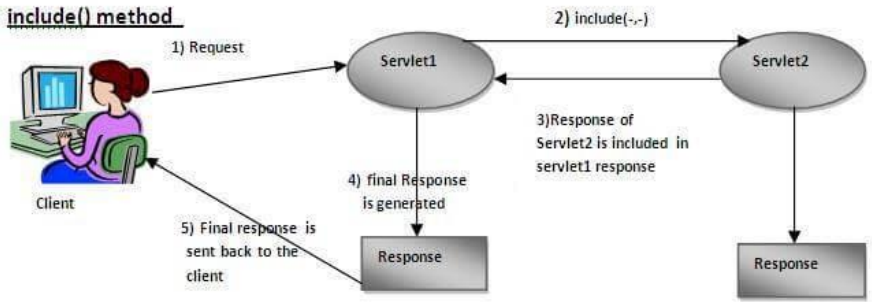
Methods of RequestDispatcher interface

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.



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

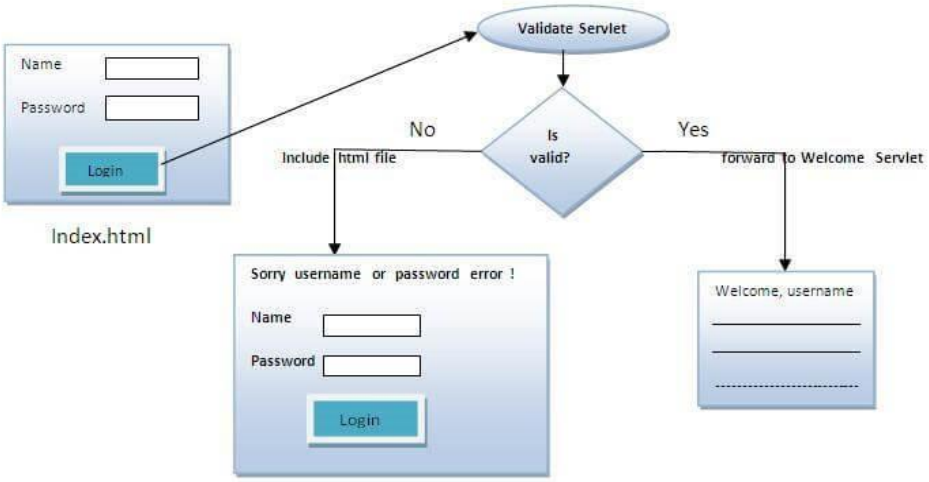
Example of using getRequestDispatcher method

```
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 servlet, 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

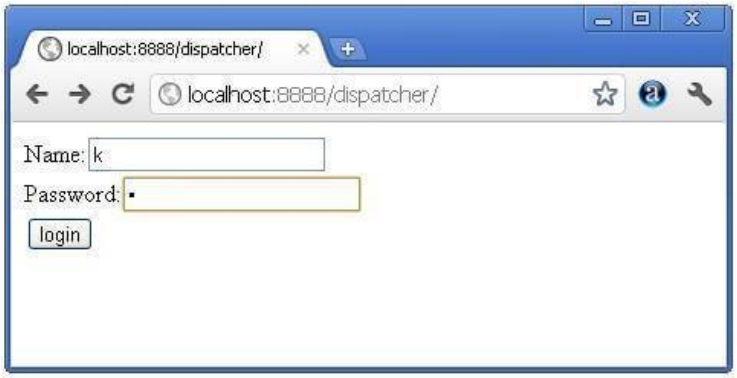
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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);
    }

}
```

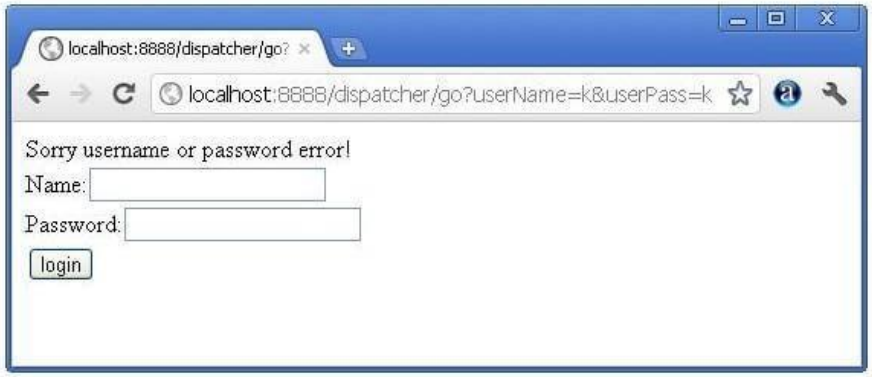
web.xml

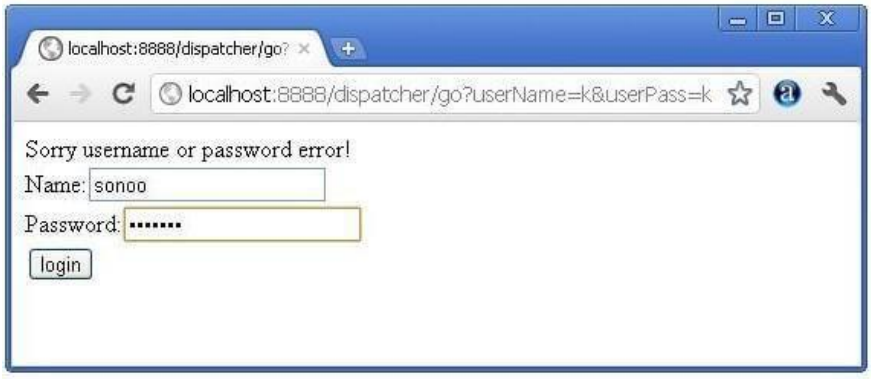
```
<web-app>
 <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>Login</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-class>WelcomeServlet</servlet-class>
  </servlet>


  <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>
```

download this example download this example (developed in Myeclipse IDE) download this example (developed in eclipse IDE) download this example (developed in netbeans IDE)








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

SendRedirect in servlet

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.getRequestDispacher("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.javatpoint.com");
```

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);
   }
}
```

Output

ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

How to get the object of ServletConfig

1. **getServletConfig() method** of Servlet interface returns the object of ServletConfig.

Syntax of getServletConfig() method

```
public ServletConfig getServletConfig();
```

Example of getServletConfig() method

```
ServletConfig config=getServletConfig();
//Now we can call the methods of ServletConfig interface
```

Syntax to provide the initialization parameter for a servlet

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

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

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

Example of ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

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

public class DemoServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   ServletConfig config=getServletConfig();
   String driver=config.getInitParameter("driver");
   out.print("Driver is: "+driver);

   out.close();
   }

}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>

<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

</web-app>
```

Example of ServletConfig to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file and printing this information in the servlet.

DemoServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;



public class DemoServlet extends HttpServlet {
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    ServletConfig config=getServletConfig();
    Enumeration<String> e=config.getInitParameterNames();

    String str="";
    while(e.hasMoreElements()){
    str=e.nextElement();
    out.print("<br>Name: "+str);
    out.print(" value: "+config.getInitParameter(str));
    }

    out.close();
}

}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>

<init-param>
<param-name>username</param-name>
<param-value>system</param-value>
</init-param>

<init-param>
<param-name>password</param-name>
<param-value>oracle</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

</web-app>
```

ServletContext Interface

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

```
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 and paramvalue 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:

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();

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

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

pw.close();

}}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>

</web-app>
```

Example of ServletContext to get all the initialization parameters

In this example, we are getting all the initialization parameter from the web.xml file. For getting all the parameters, we have used the getInitParameterNames() method in the servlet class.

DemoServlet.java

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


public class DemoServlet extends HttpServlet{
```

```
 publicvoid doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();

ServletContext context=getServletContext();
Enumeration<String> e=context.getInitParameterNames();

String str="";
while(e.hasMoreElements()){
   str=e.nextElement();
   out.print("<br> "+context.getInitParameter(str));
}
}}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

<context-param>
<param-name>username</param-name>
<param-value>system</param-value>
</context-param>

<context-param>
<param-name>password</param-name>
<param-value>oracle</param-value>
</context-param>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>

</web-app>
```

Attribute in Servlet

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

The servlet programmer can pass informations from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

Attribute specific methods of ServletRequest, HttpSession and ServletContext interface

There are following 4 attribute specific methods. They are as follows:

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

Example of ServletContext to set and get attribute

In this example, we are setting the attribute in the application scope and getting that value from another servlet.

DemoServlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet1 extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
{
try{

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


ServletContext context=getServletContext();
context.setAttribute("company","IBM");

out.println("Welcome to first servlet");
out.println("<a href='servlet2'>visit</a>");
out.close();

}catch(Exception e){out.println(e);}

}}
```

DemoServlet2.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DemoServlet2 extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
{
try{

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

ServletContext context=getServletContext();
String n=(String)context.getAttribute("company");

out.println("Welcome to "+n);
out.close();

}catch(Exception e){out.println(e);}
}}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>DemoServlet1</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>DemoServlet2</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

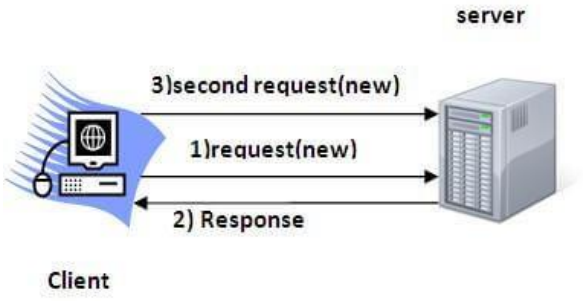
Session Tracking in Servlets

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**

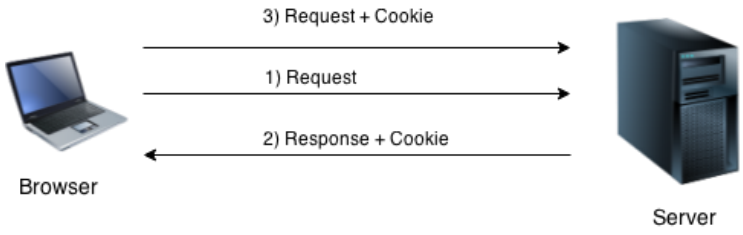
Cookies in Servlet

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.

> Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.

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","sonoo jaiswal");//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
```

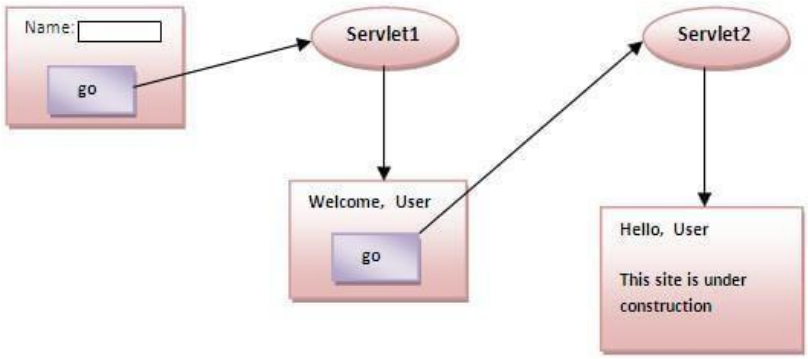
How to get Cookies?

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
}
```

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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



public class FirstServlet extends HttpServlet {

  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    String n=request.getParameter("userName");
    out.print("Welcome "+n);

    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response

    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");

    out.close();

        }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

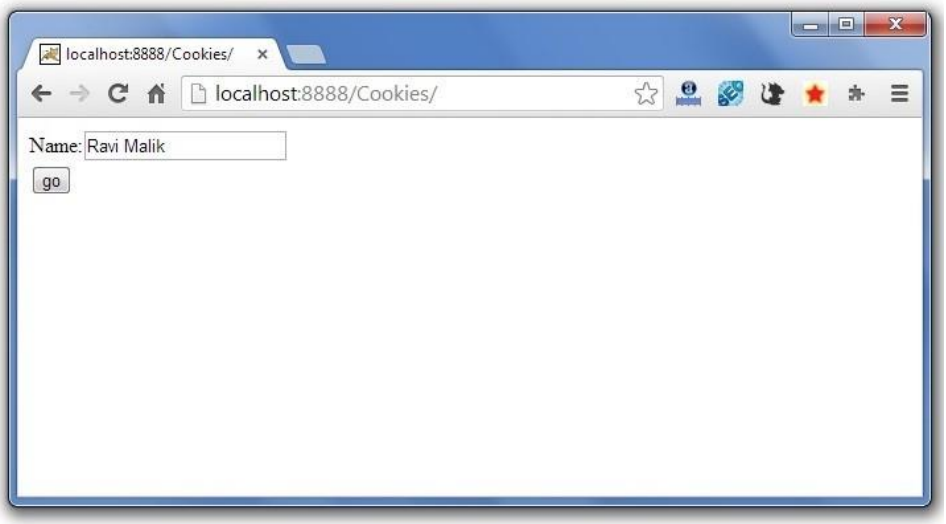
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response){
   try{

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   Cookie ck[]=request.getCookies();
   out.print("Hello "+ck[0].getValue());

   out.close();

       }catch(Exception e){System.out.println(e);}
   }


}
```

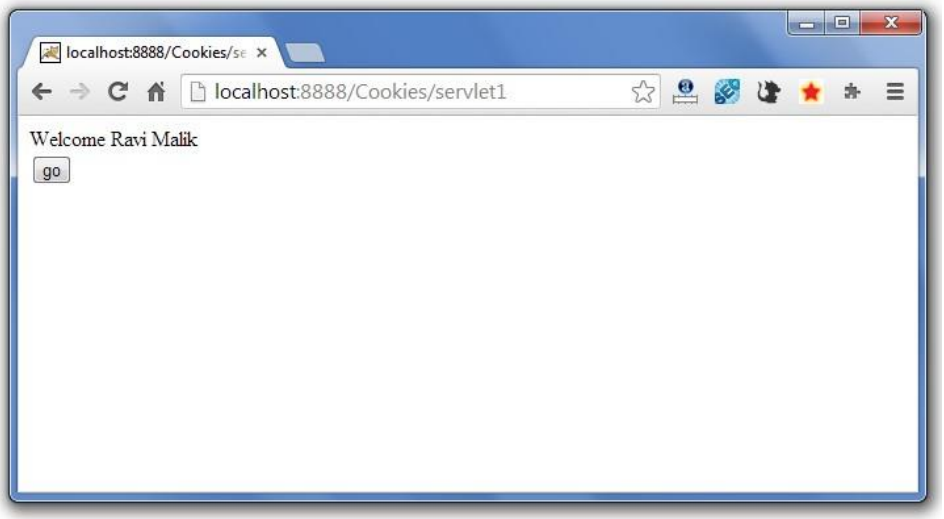
web.xml

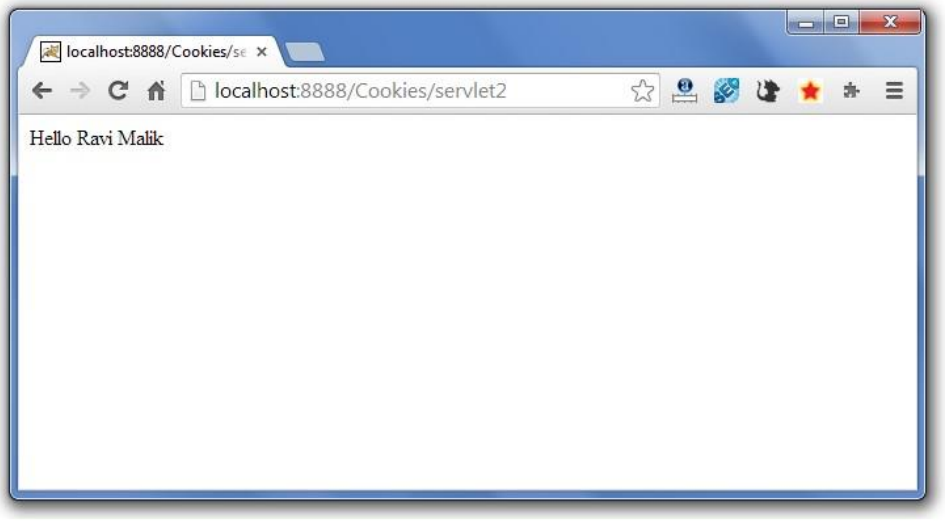
```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

download this example (developed using Myeclipse IDE) download this example (developed using Eclipse IDE) download this example (developed using Netbeans IDE)

Output







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

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>

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

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

File: link.html

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

File: login.html

```
<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>
```

File: LoginServlet.java

```
package com.javatpoint;

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();
   }
```

```
}
```

File: LogoutServlet.java

```
package com.javatpoint;

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!");
   }
}
```

File: ProfileServlet.java

```
package com.javatpoint;

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();
   }
}
```

File: web.xml

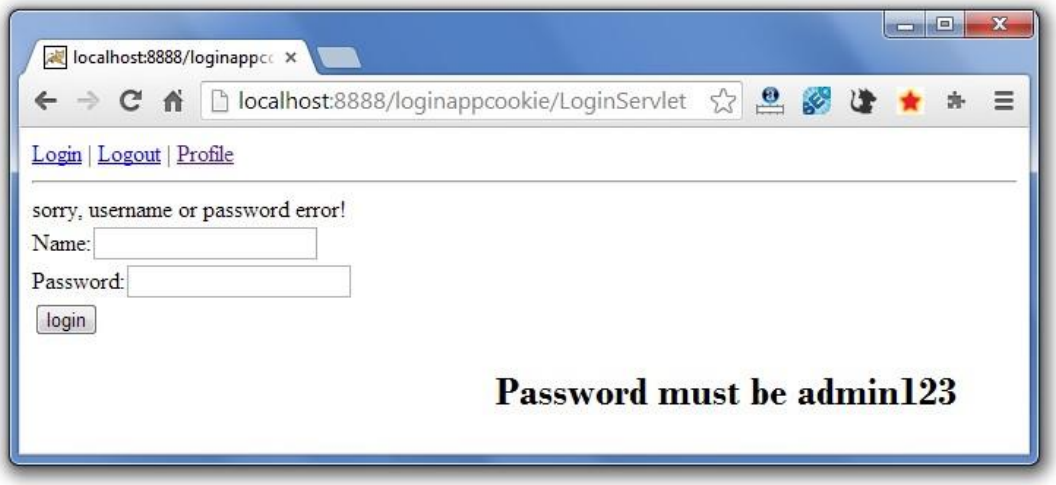
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

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

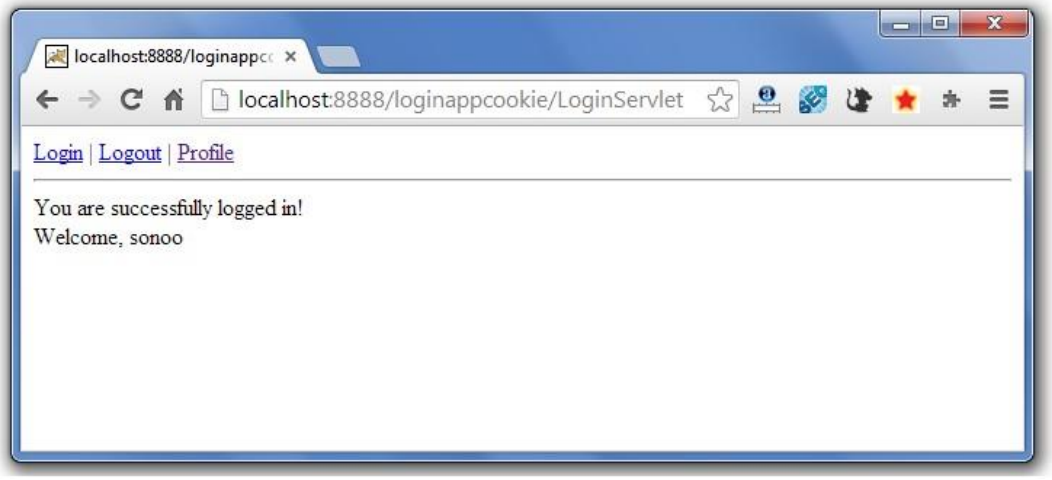
download this example (developed using Eclipse IDE)

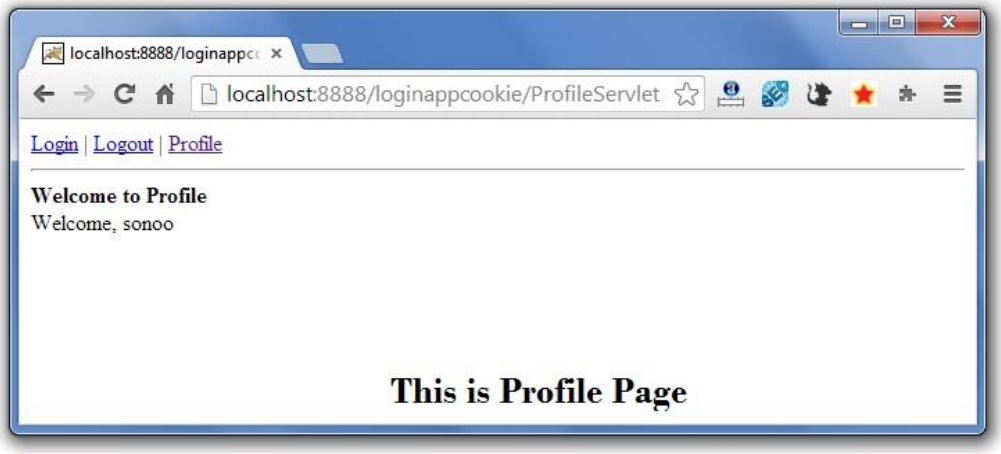
Output


Servlet Login Example
localhost:8888/loginappcookie/
Welcome to Login App by Cookie
Login| Logout| Profile


localhost:8888/loginappc
localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Please login first
Name:
Password:
login
You can't visit profile page directly


localhost:8888/loginappc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
sorry, username or password error!
Name:
Password:
login
Password must be admin123


localhost:8888/loginappc
localhost:8888/loginappcookie/LoginServlet
Login | Logout | Profile
You are successfully logged in!
Welcome, sonoo



If again you click on the profile link, you need to login first.

2) 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.

```
<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

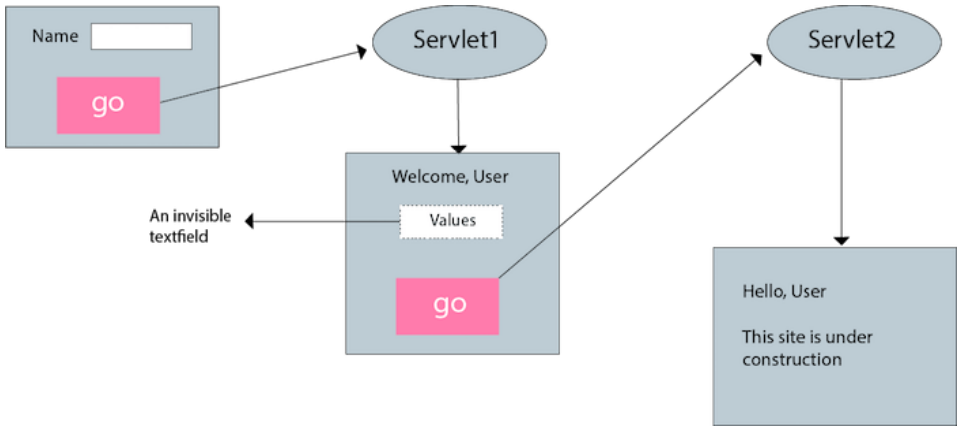
1. It will always work whether cookie is disabled or not.

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.



index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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

public class FirstServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      //creating form that have invisible textfield
      out.print("<form action='servlet2'>");
      out.print("<input type='hidden' name='uname' value='"+n+"'>");
      out.print("<input type='submit' value='go'>");
      out.print("</form>");
      out.close();

            }catch(Exception e){System.out.println(e);}
   }

}
```

SecondServlet.java

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

```
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //Getting the value from the hidden field
      String n=request.getParameter("uname");
      out.print("Hello "+n);

      out.close();
            }catch(Exception e){System.out.println(e);}
   }
}
```

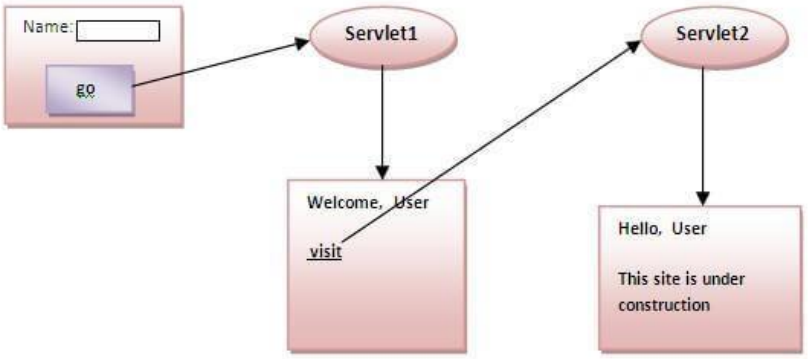
web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

3)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 maintaning 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.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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


public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      //appending the username in the query string
      out.print("<a href='servlet2?uname="+n+"'>visit</a>");

      out.close();

            }catch(Exception e){System.out.println(e);}
   }

}
```

SecondServlet.java

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      //getting value from the query string
      String n=request.getParameter("uname");
      out.print("Hello "+n);

      out.close();

            }catch(Exception e){System.out.println(e);}
   }


}
```

web.xml

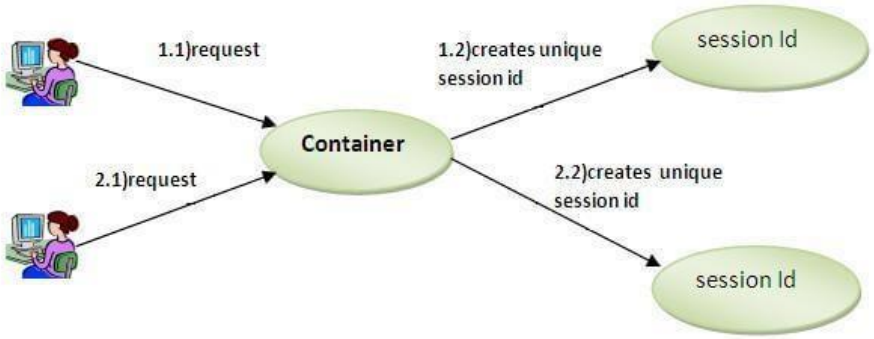
```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```

download this example (developed using Myeclipse IDE) download this example (developed using Eclipse IDE) download this example (developed using Netbeans IDE)

4) HttpSession interface

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.

index.html

```
<form action="servlet1">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

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



public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("userName");
      out.print("Welcome "+n);

      HttpSession session=request.getSession();
      session.setAttribute("uname",n);

      out.print("<a href='servlet2'>visit</a>");

      out.close();

            }catch(Exception e){System.out.println(e);}
   }

}
```

SecondServlet.java

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

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

public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
      try{

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      HttpSession session=request.getSession(false);
      String n=(String)session.getAttribute("uname");
      out.print("Hello "+n);

      out.close();

            }catch(Exception e){System.out.println(e);}
   }


}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>

</web-app>
```