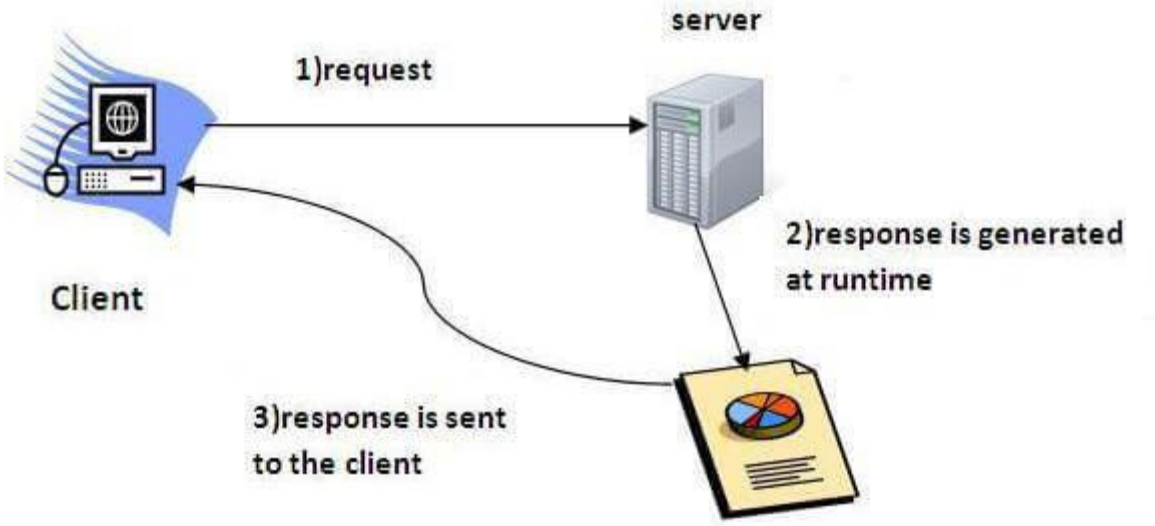
Java Servlet

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.

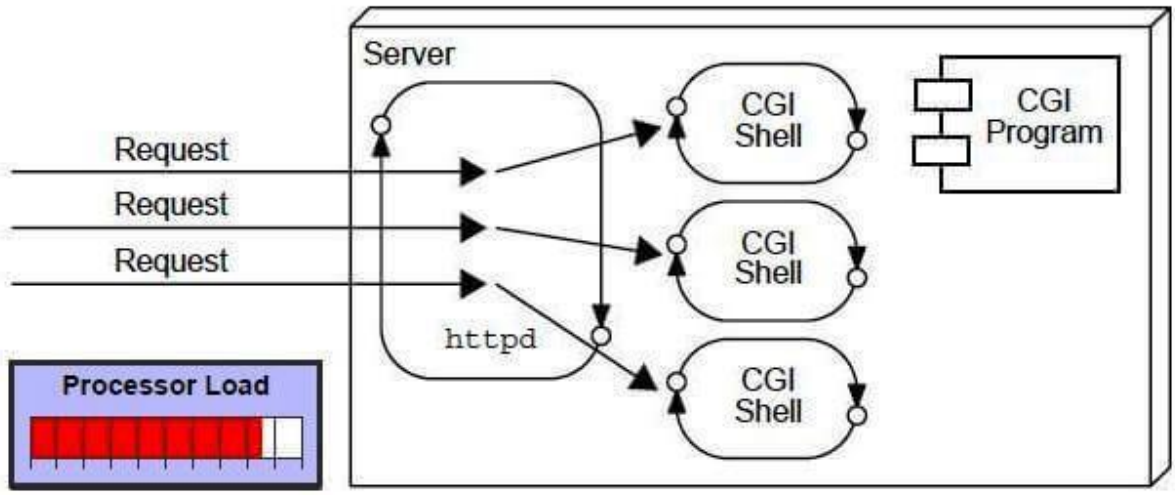


What is a web application?

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.

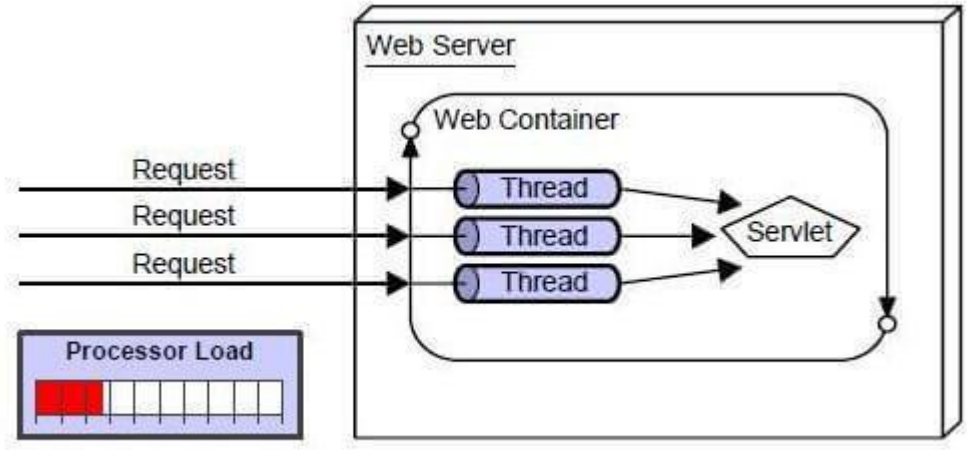


Disadvantages of CGI

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



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.

Servlet API

1. Servlet API
2. Interfaces in javax.servlet package
3. Classes in javax.servlet package
4. Interfaces in javax.servlet.http package
5. Classes in javax.servlet.http package

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.

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

Classes in javax.servlet package

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

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletException
5. 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

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

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.

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 Program

```
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>");
  }
}
```

HttpServlet class

1. HttpServlet class

2. Methods of HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It prov
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.

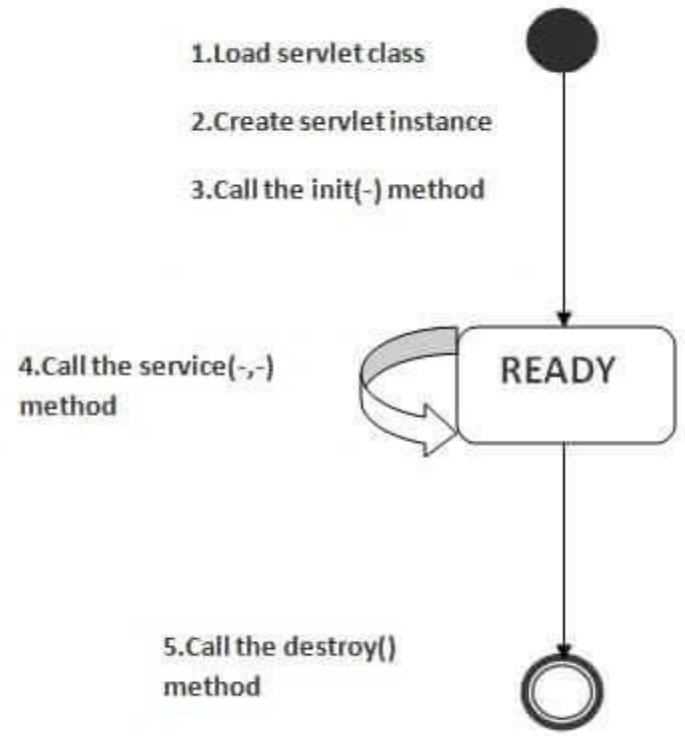
Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a 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

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.

1) Servlet class is loaded

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 metho
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

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

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

The servlet example can be created by three ways:

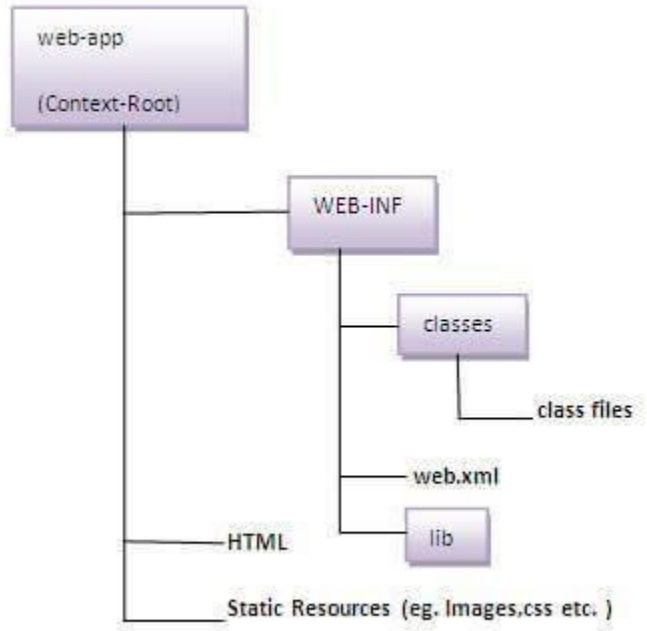
1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

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

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.



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 doHead() etc.

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

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>
```

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).

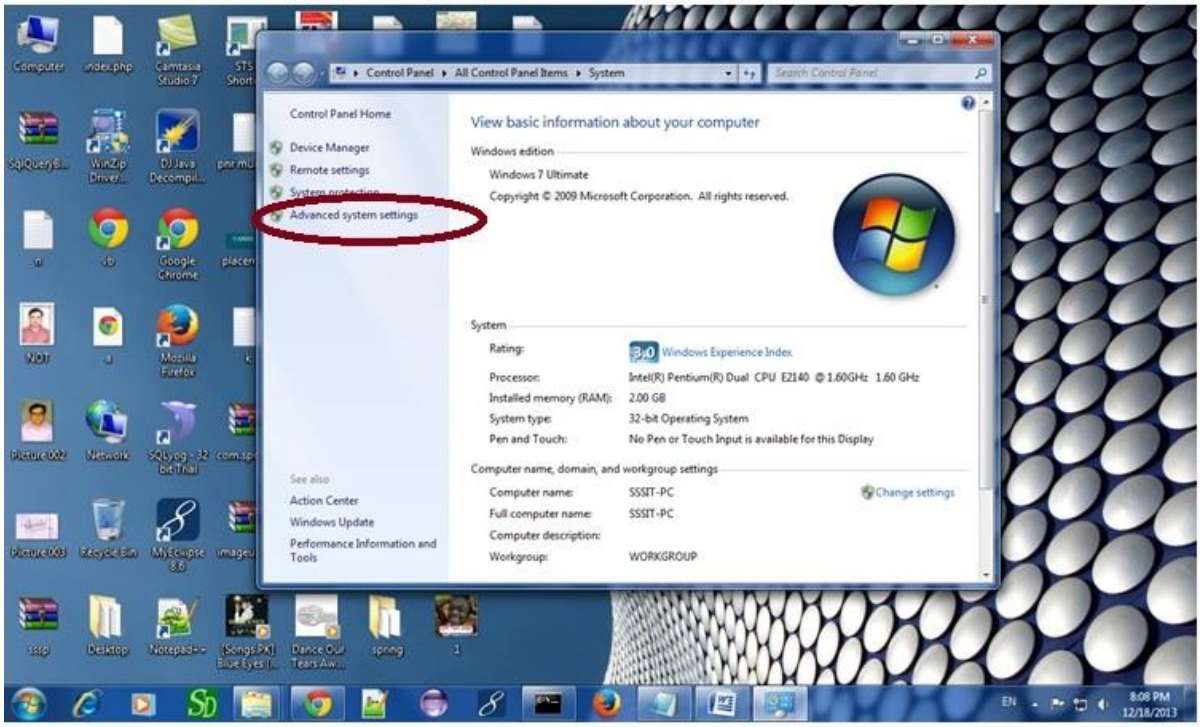
1) How to set JAVA_HOME in environment variable?

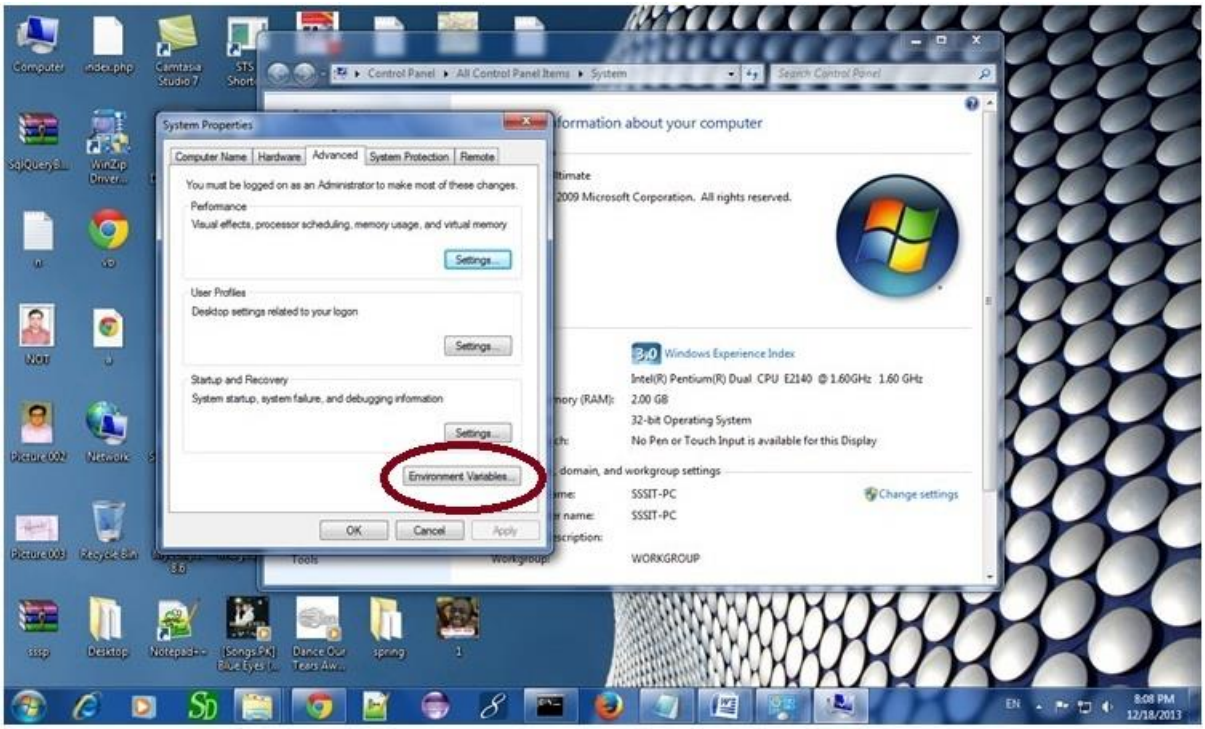
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

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:

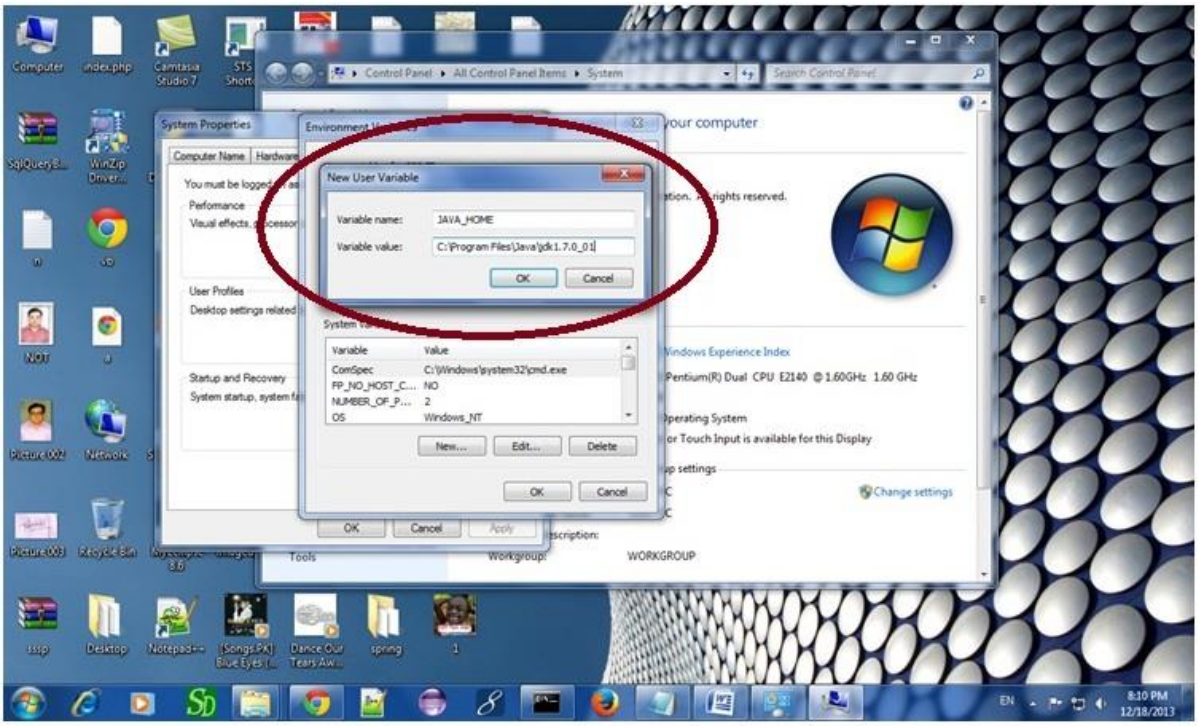
Click on advanced system settings tab then environment variables:





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:



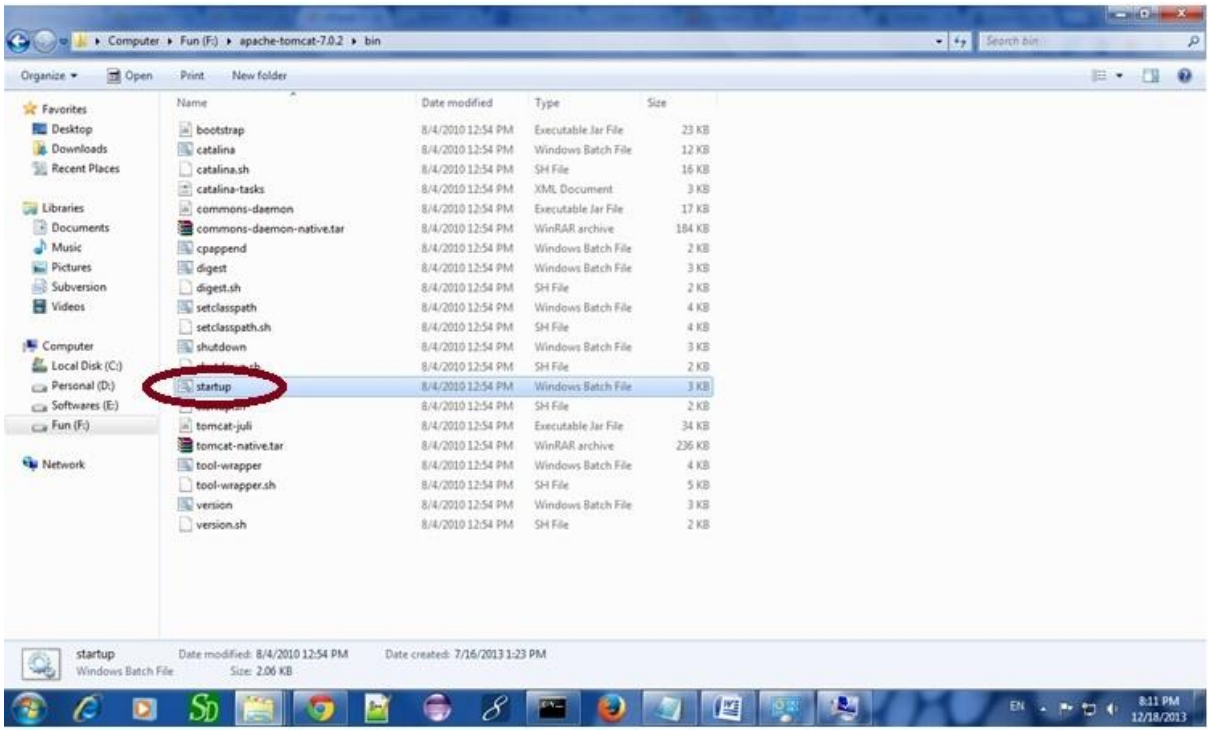
There must not be semicolon (;) at the end of the path.

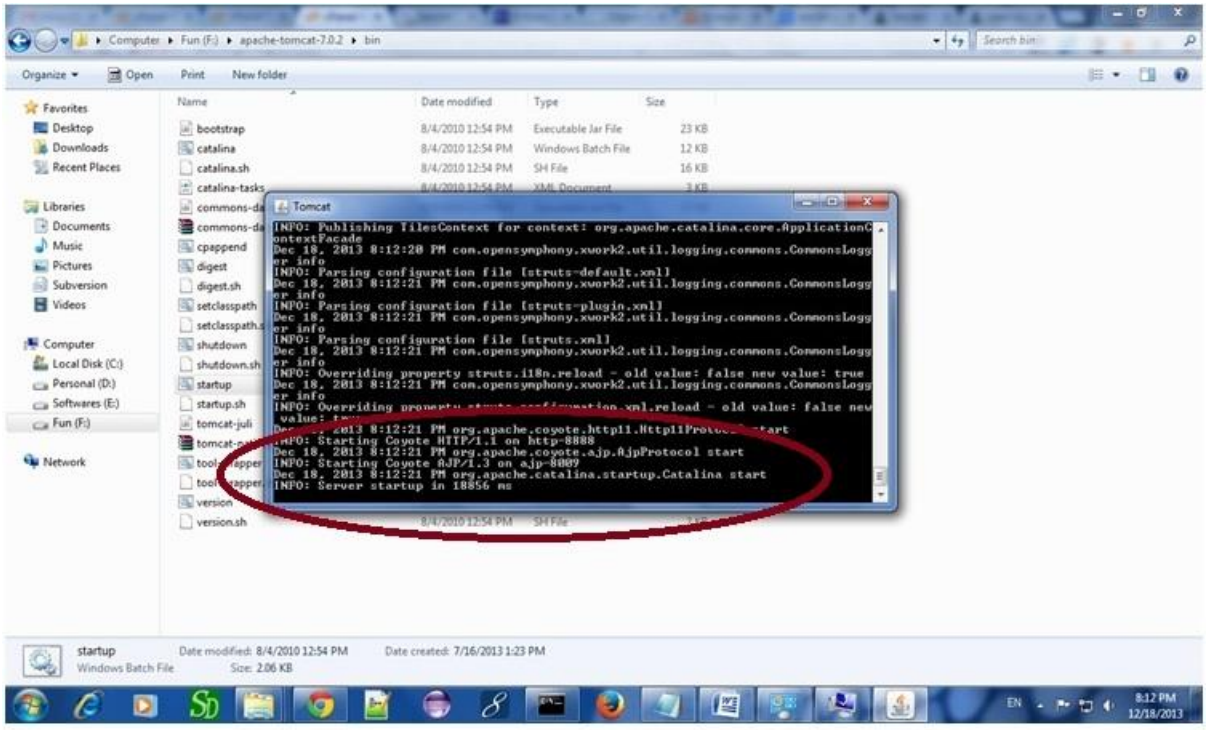
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

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

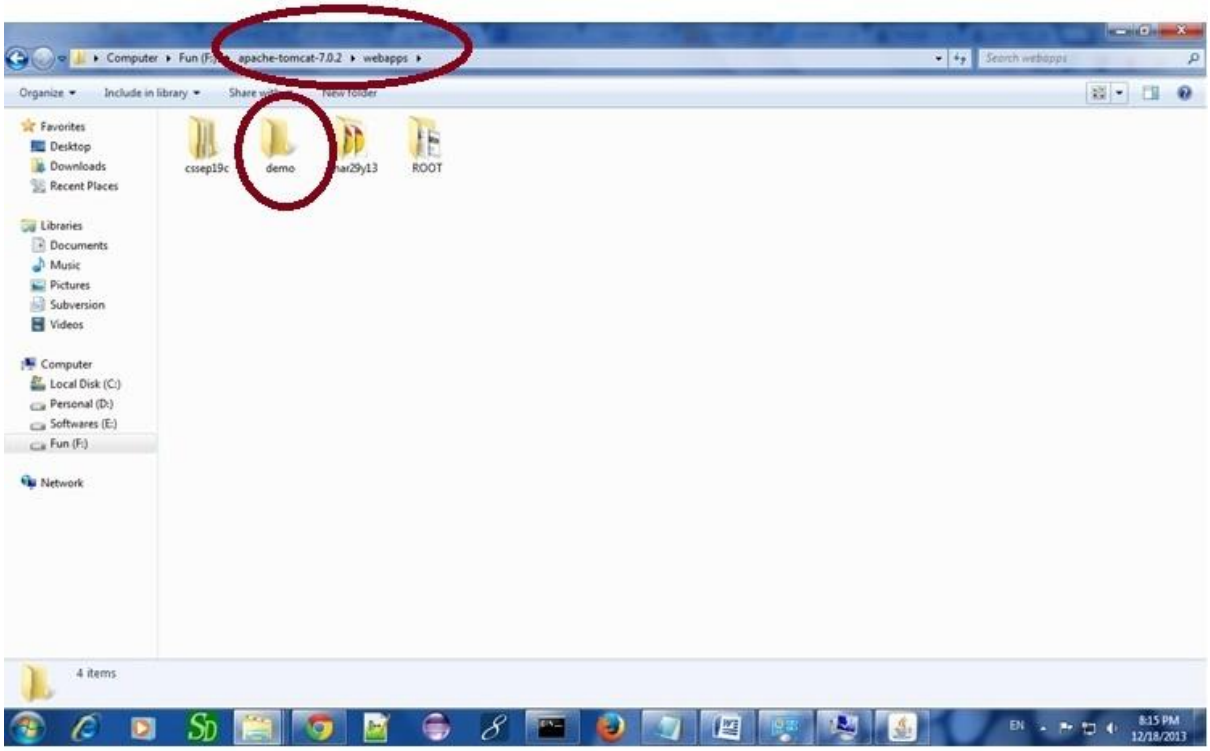
Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

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:

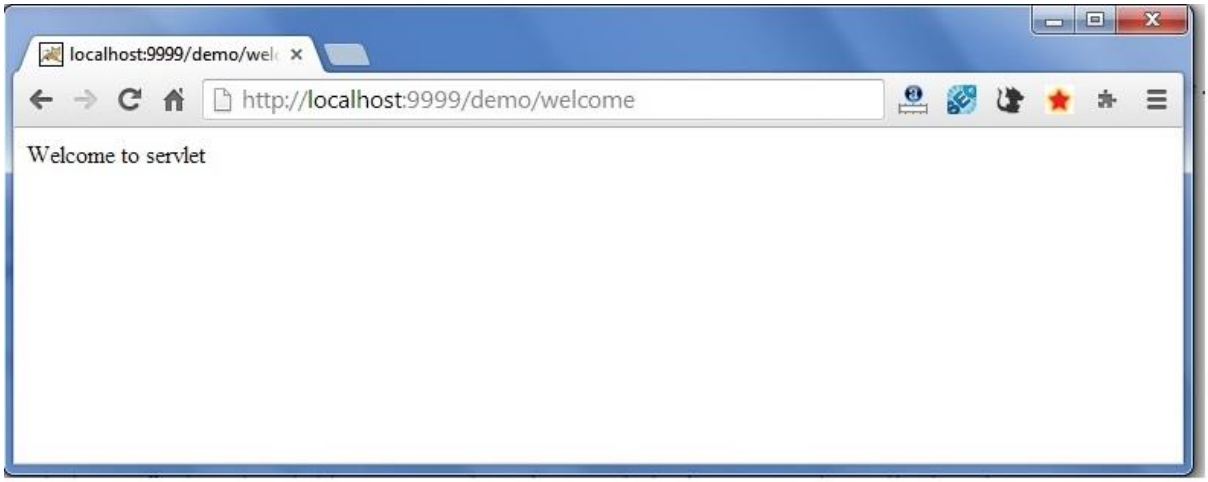
1. projectfolder> jar cvf myproject.war *

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:

1. http://localhost:9999/demo/welcome



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/.

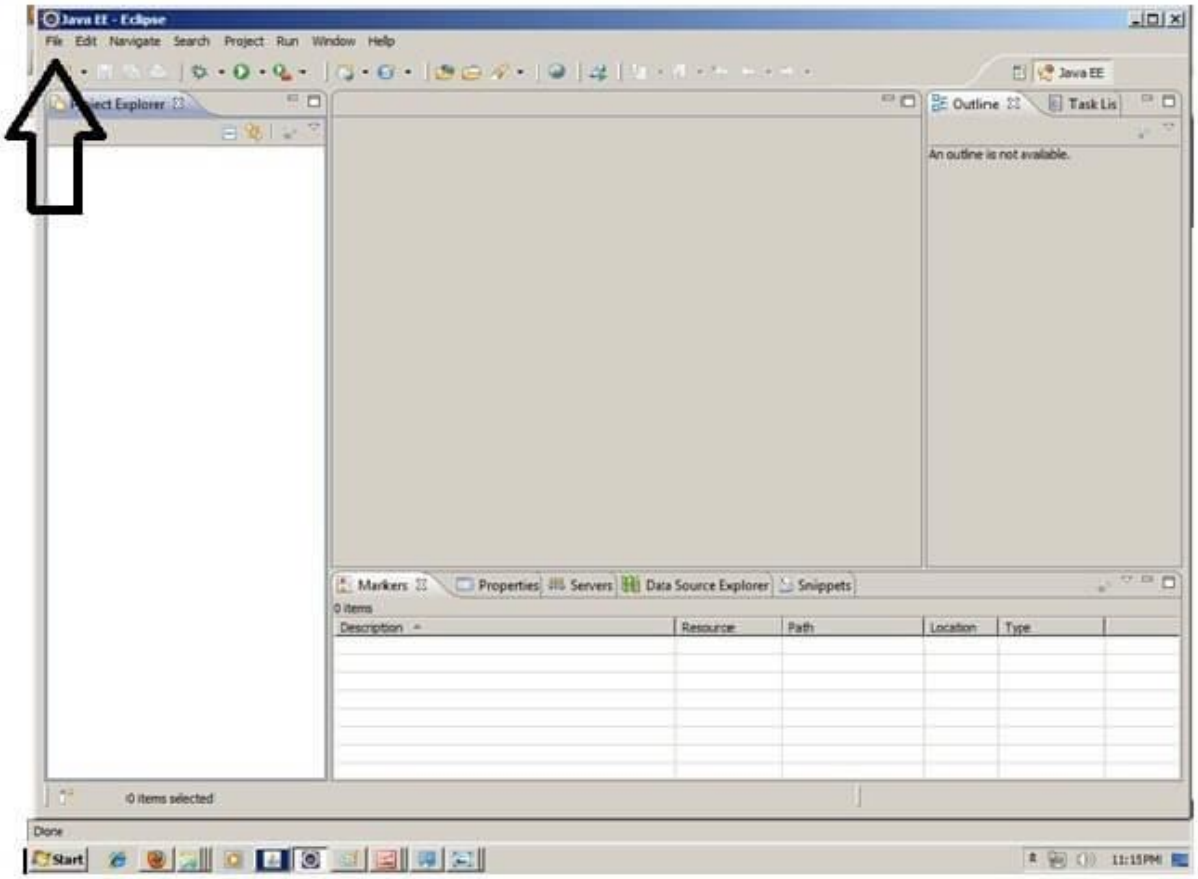
You need to download the eclipse ide for JavaEE developers.

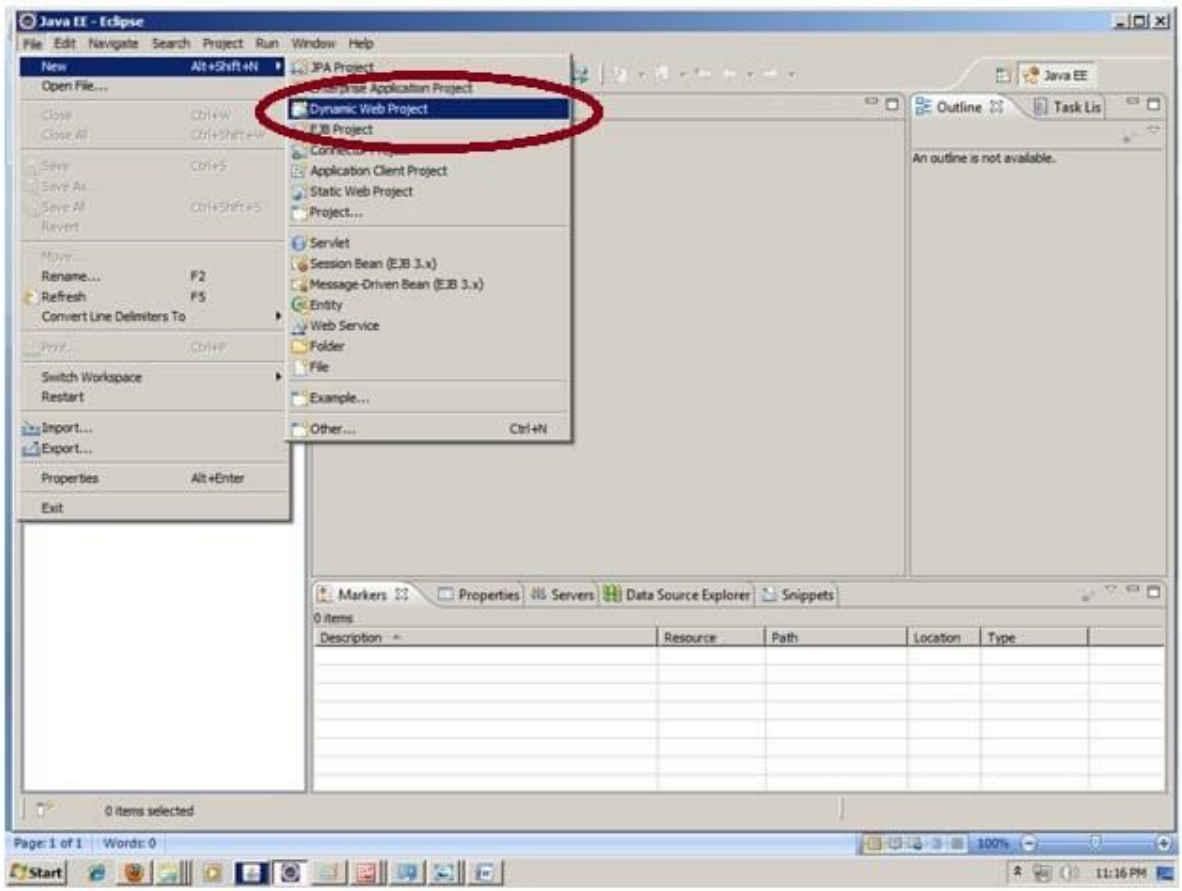
Creating **servlet example in eclipse ide**, saves a lot of work to be done. It is easy and simple to create a servlet example. Let's see the steps, you need to follow to create the first servlet example.

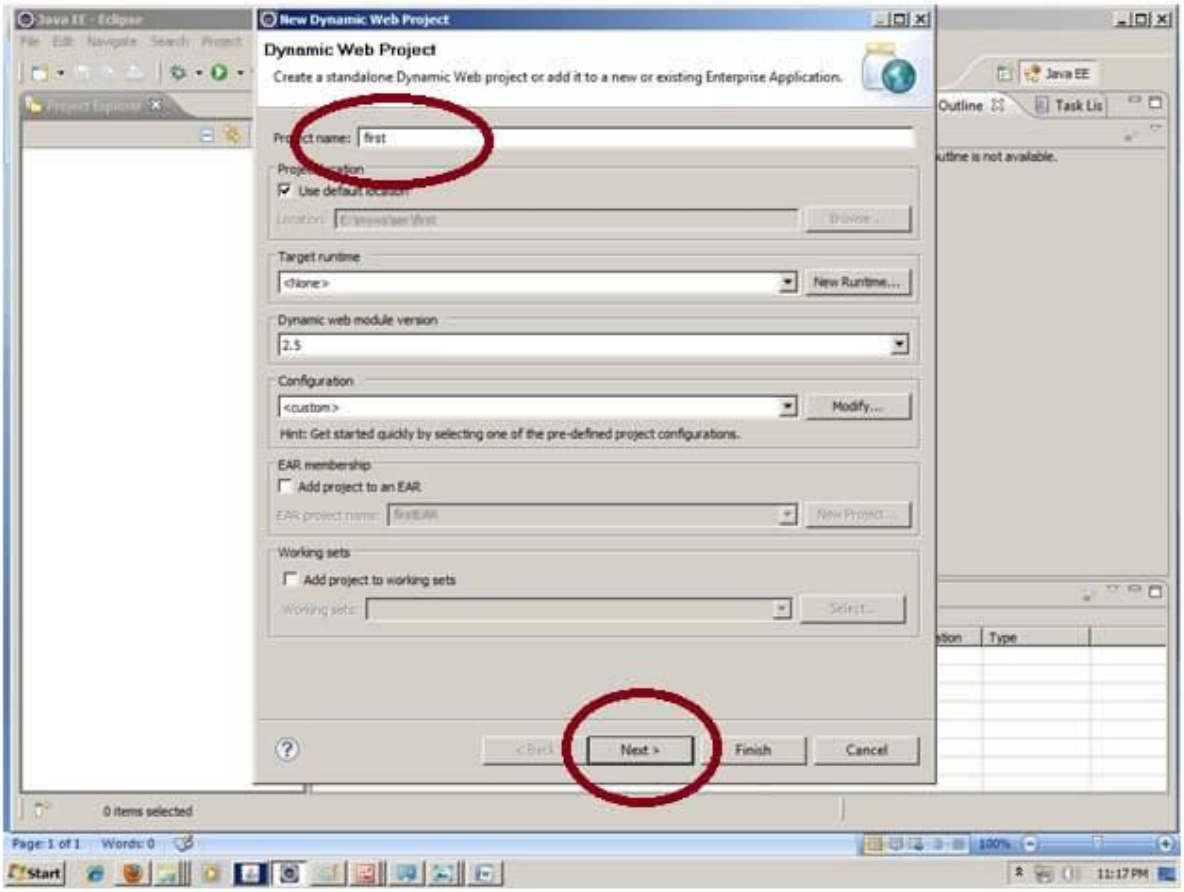
- Create a Dynamic web project
- create a servlet
- add servlet-api.jar file
- Run the servlet

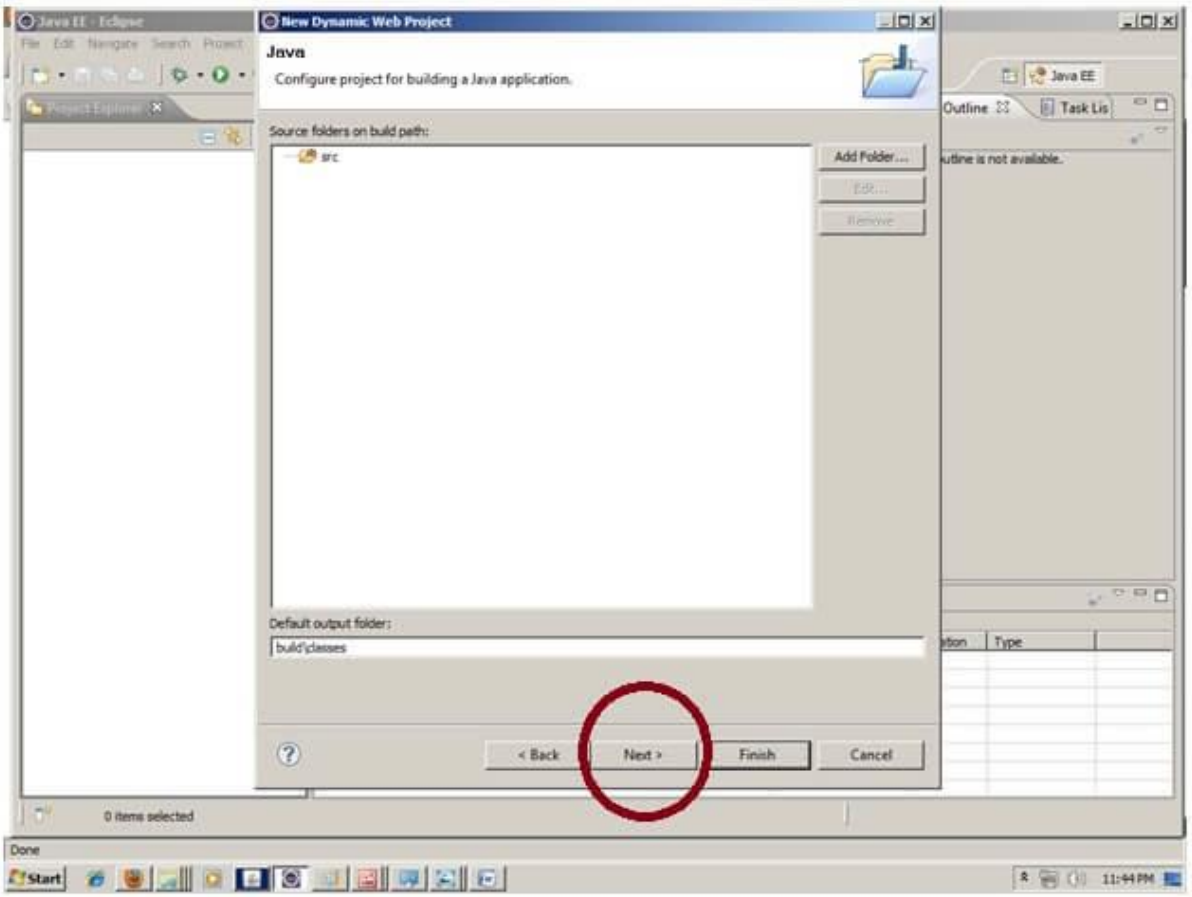
1) Create the dynamic web project:

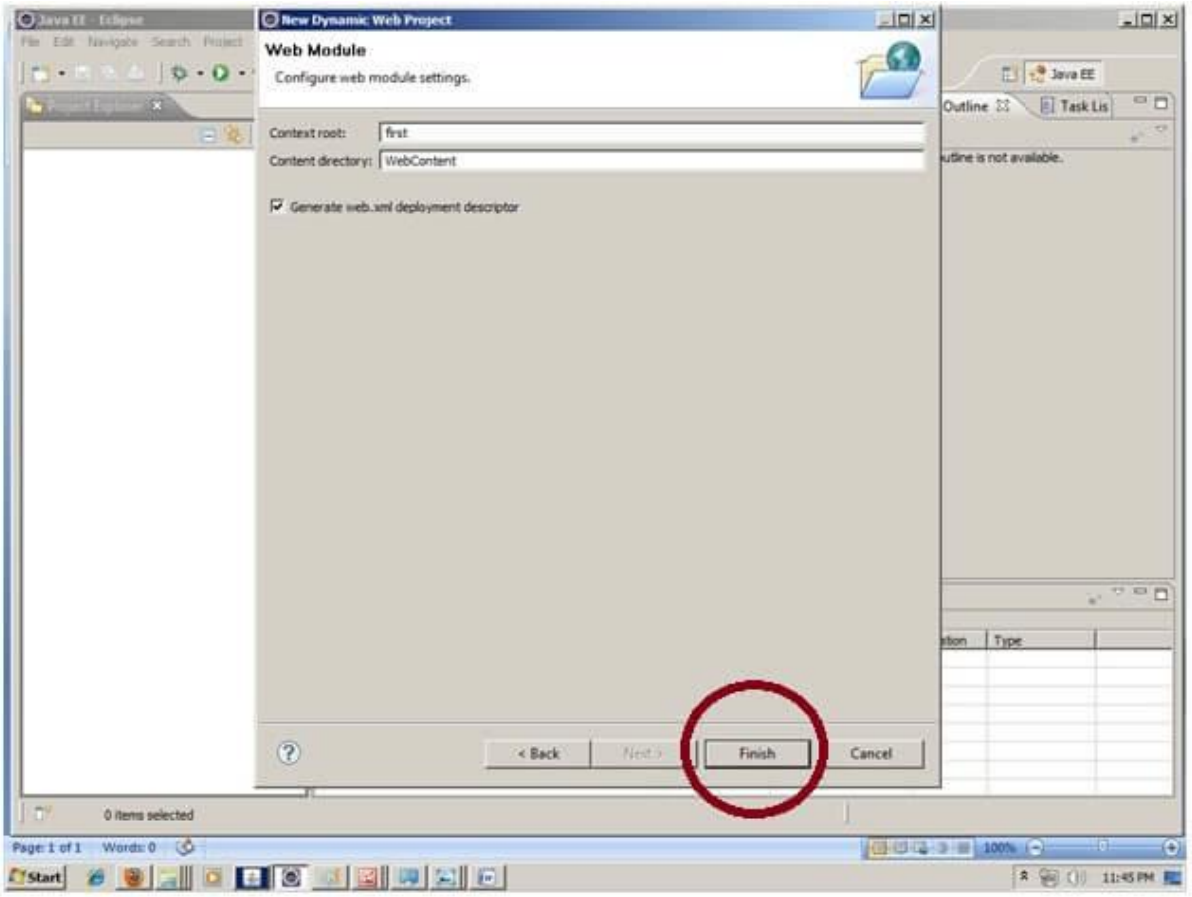
For creating a dynamic web project **click on File Menu -> New -> Project..-> Web -> dynamic web project -> write your project name e.g. first -> Finish**.

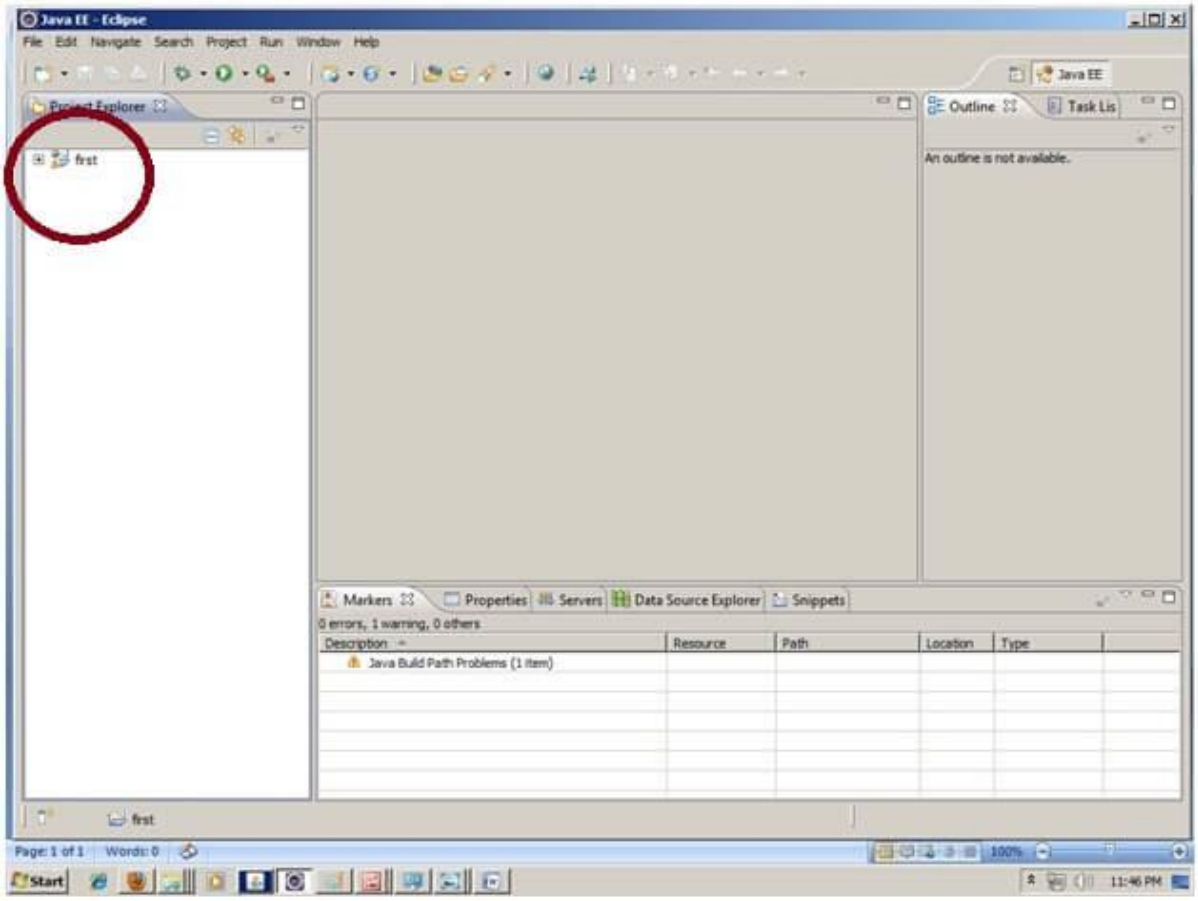

Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Java EE
Outline
Task Lis
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Description
Resource
Path
Location
Type
0 items selected
Done
Start
11:15 PM


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
Enterprise Application Project
Dynamic Web Project
EJB Project
Connector 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
Java EE
Outline
Task Lis
An outline is not available.
Markers
Properties
Servers
Data Source Explorer
Snippets
0 items
Description
Resource
Path
Location
Type
0 items selected
Page: 1 of 1
Words: 0
100%
Start
11:16 PM


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
Next >
Finish
Cancel
0 items selected

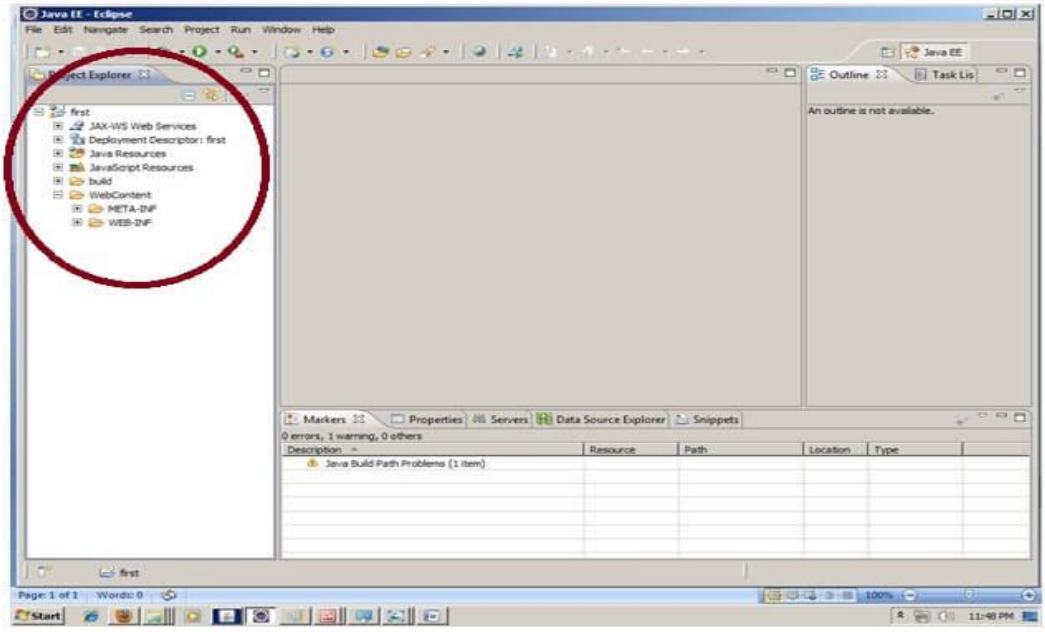

New Dynamic Web Project
Java
Configure project for building a Java application.
Source folders on build path:
src
Add Folder...
Edit...
Remove
Default output folder:
build\classes
< Back
Next >
Finish
Cancel
Java EE
Outline
Task Lis
utline is not available.
Type
0 items selected
Done
Start
11:44 PM

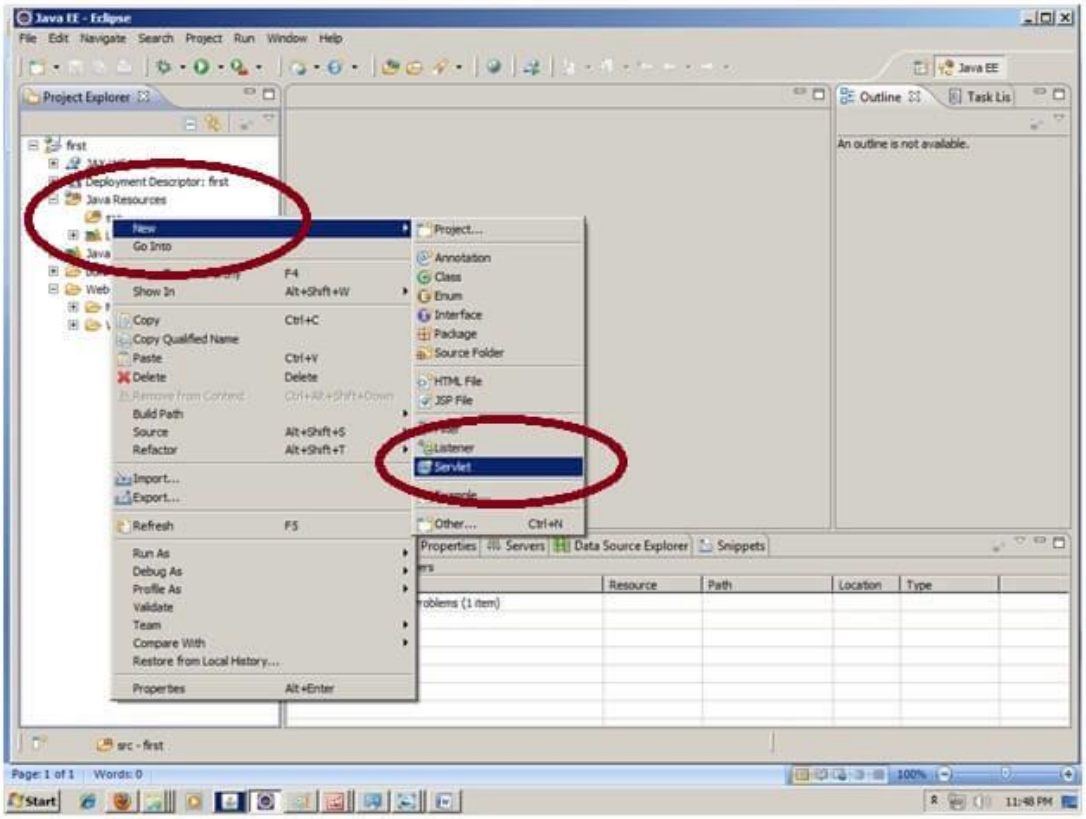

New Dynamic Web Project
Web Module
Configure web module settings.
Context root:
first
Content directory:
WebContent
Generate web.xml deployment descriptor
< Back
Next >
Finish
Cancel
Java EE
Outline
Task Lis
utline is not available.
Type
0 items selected
Page: 1 of 1
Words: 0
100%
Start
11:45 PM

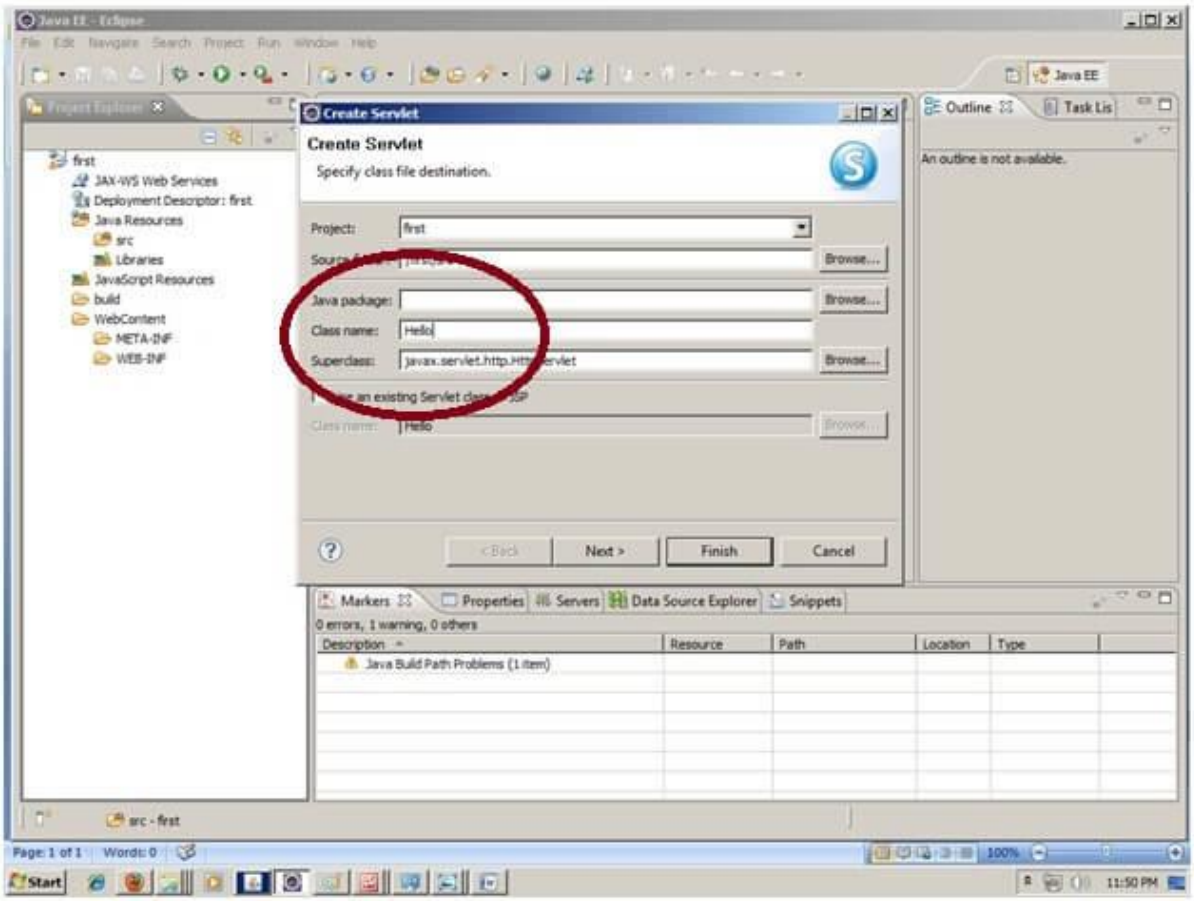


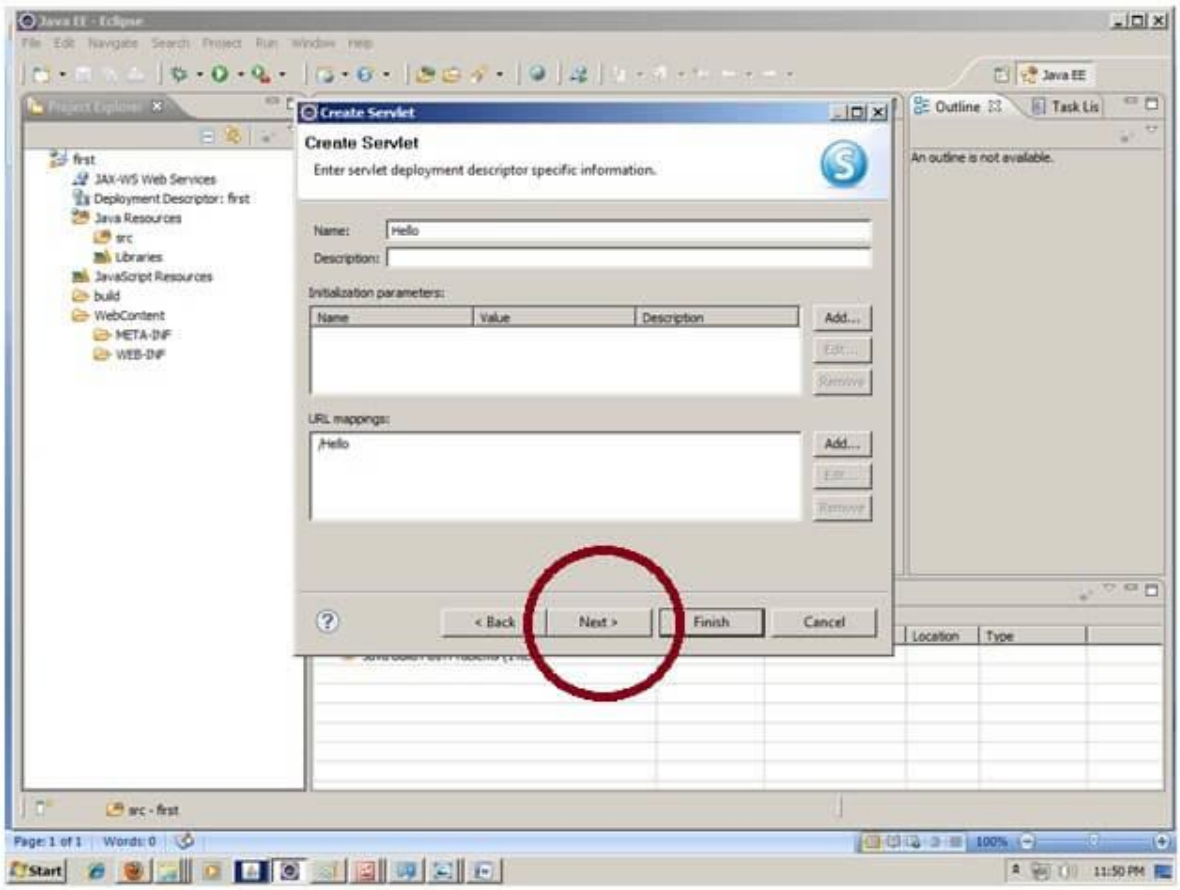
2) Create the servlet in eclipse IDE:

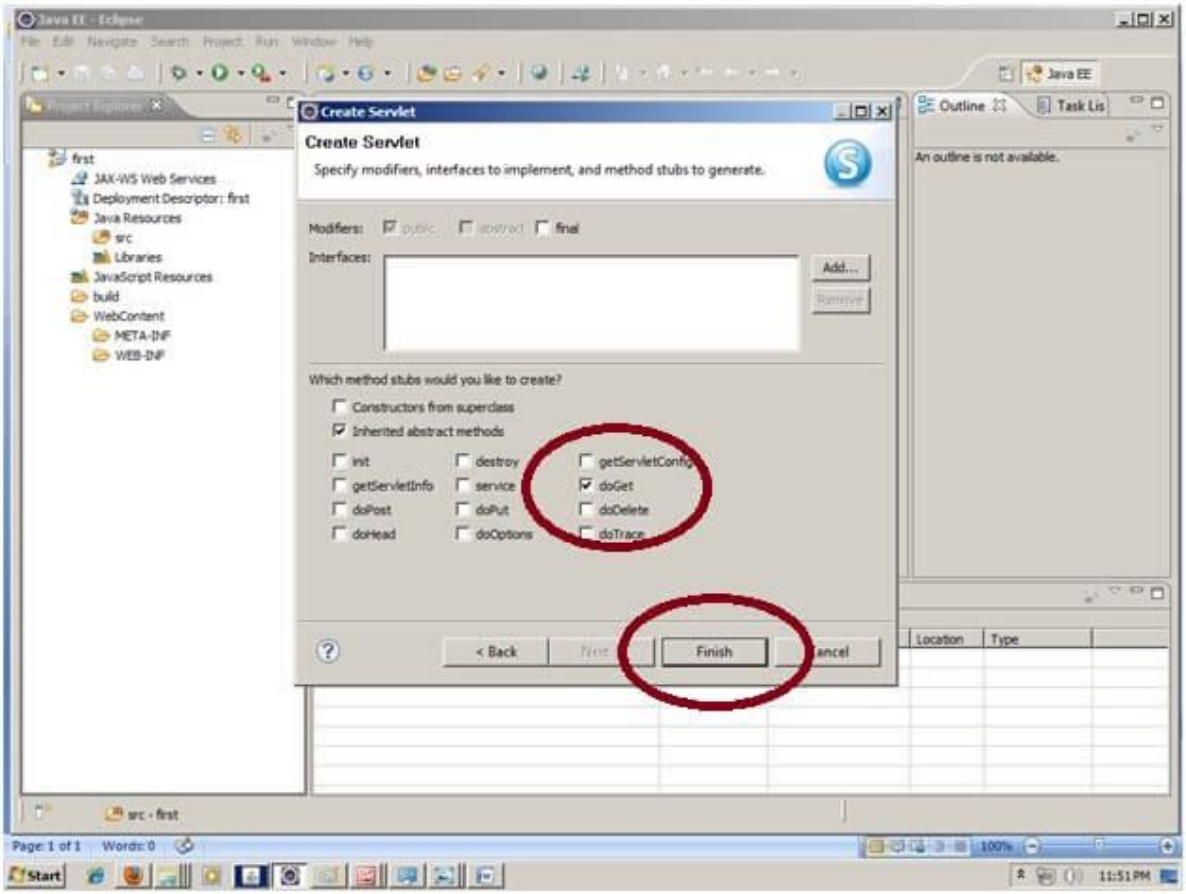
For creating a servlet, **explore the project by clicking the + icon -> explore the Java Resources -> right click on src -> New -> servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish**.

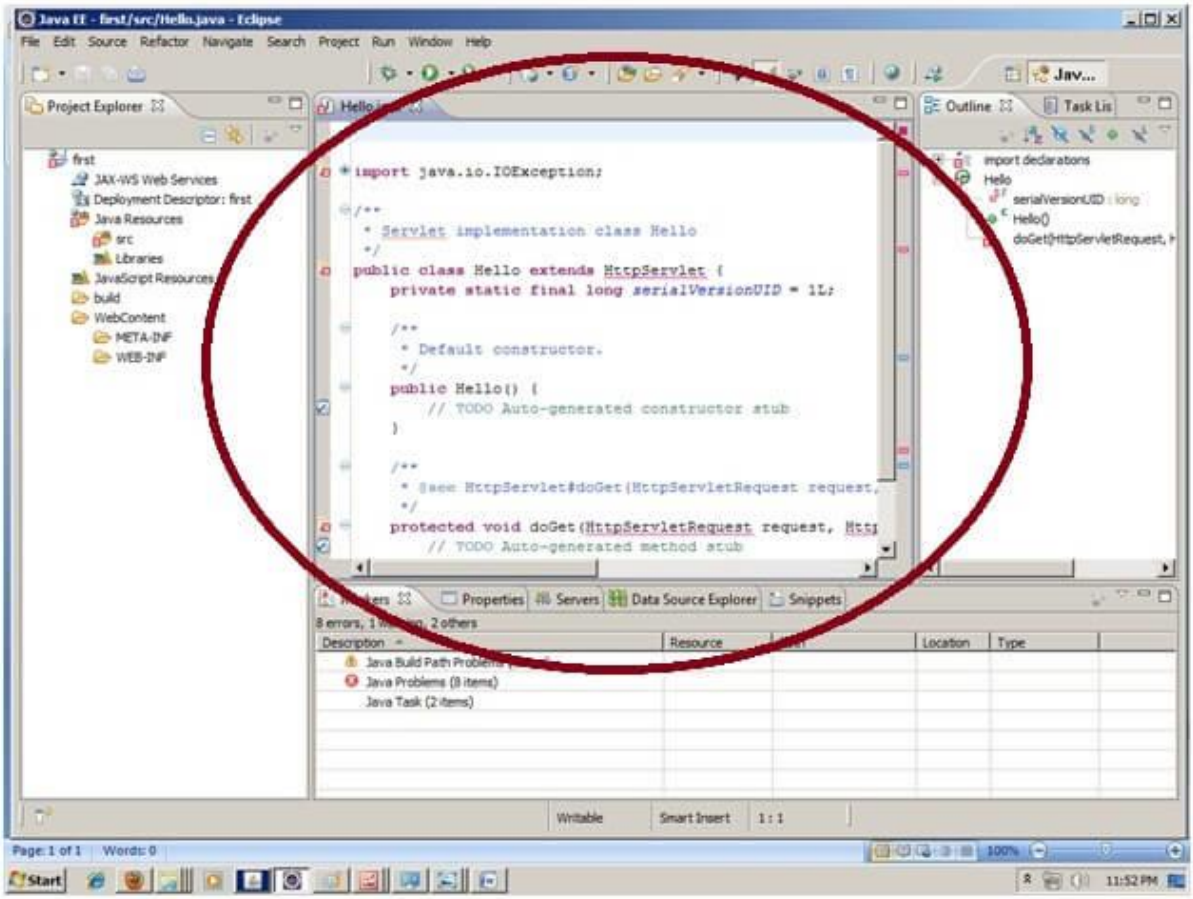



Java EE - Eclipse
File Edit Navigate Search Project Run Window Help
Project Explorer
first
Deployment Descriptor: first
Java Resources
New
Go Into
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Run As
Debug As
Profile As
Validate
Team
Compare With
Restore from Local History...
Properties Alt+Enter
Project...
Annotation
Class
Enum
Interface
Package
Source Folder
HTML File
JSP File
Listener
Servlet
Other... Ctrl+N
Outline
Task List
An outline is not available.
Properties
Servers
Data Source Explorer
Snippets
Resource
Path
Location
Type
Problems (1 item)
src - first
Page: 1 of 1
Words: 0
100%
Start
11:48 PM


Java EE - Eclipse
Create Servlet
Create Servlet
Specify class file destination.
Project: first
Source folder:
Java package:
Class name: Hello
Superclass: javax.servlet.http.HttpServlet
Browse...
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
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)
src - first
Page: 1 of 1
Words: 0
100%
Start
11:50 PM


Create Servlet
Create Servlet
Enter servlet deployment descriptor specific information.
Name: Hello
Description:
Initialization parameters:
Name
Value
Description
Add...
Edit...
Remove
URL mappings:
/Hello
Add...
Edit...
Remove
< Back
Next >
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
Task List
An outline is not available.
Location
Type

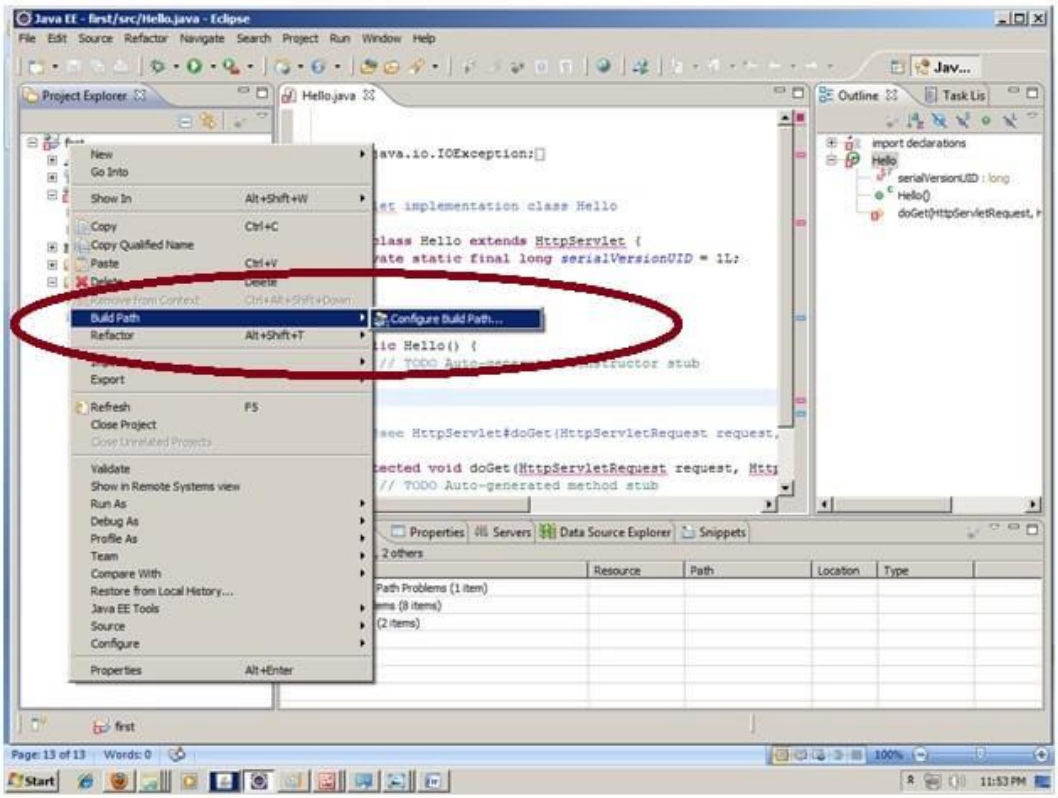

Create Servlet
Create Servlet
Specify modifiers, interfaces to implement, and method stubs to generate.
Modifiers: public abstract final
Interfaces:
Add...
Remove
Which method stubs would you like to create?
Constructors from superclass
Inherited abstract methods
init
destroy
getServletConfig
getServletInfo
service
doGet
doPost
doPut
doDelete
doHead
doOptions
doTrace
< Back
Next
Finish
Cancel
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
Outline
Task List
An outline is not available.
Location
Type

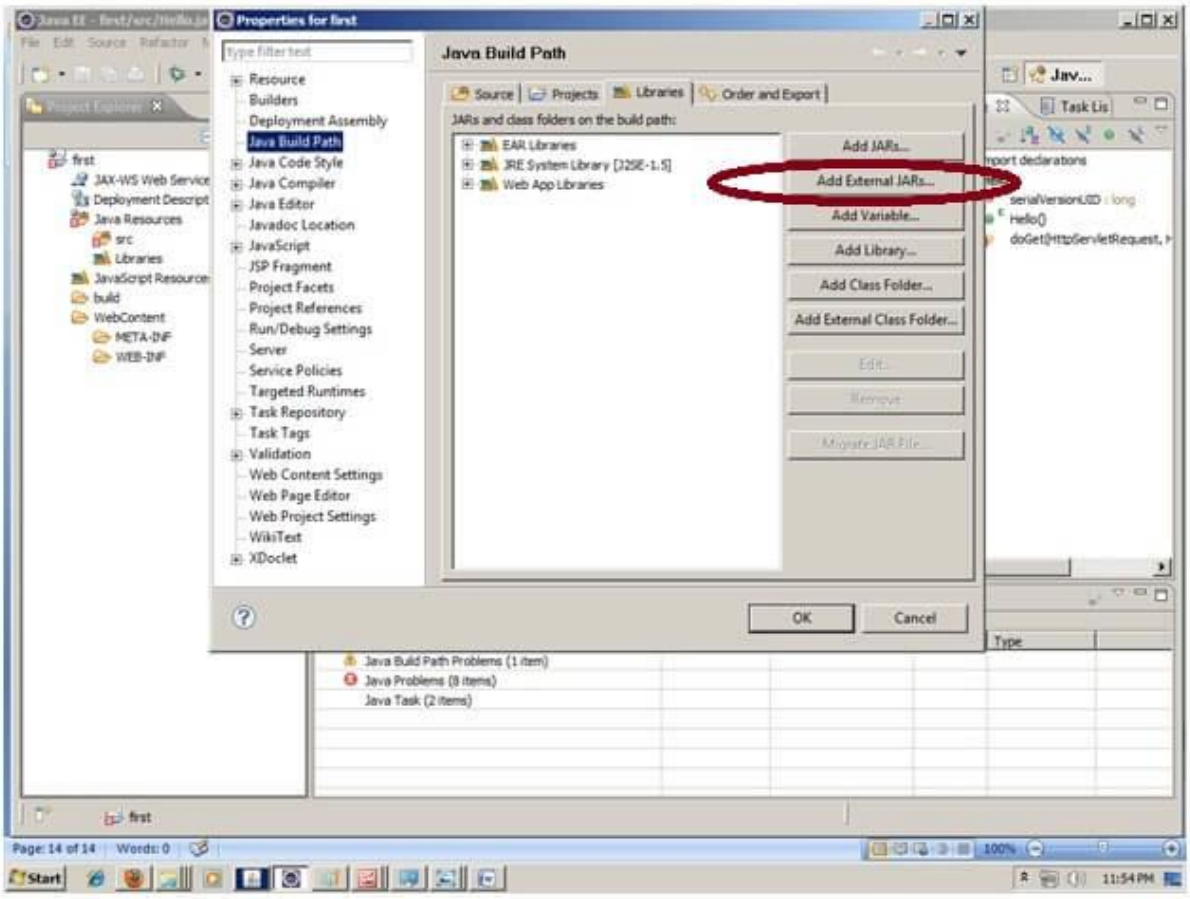


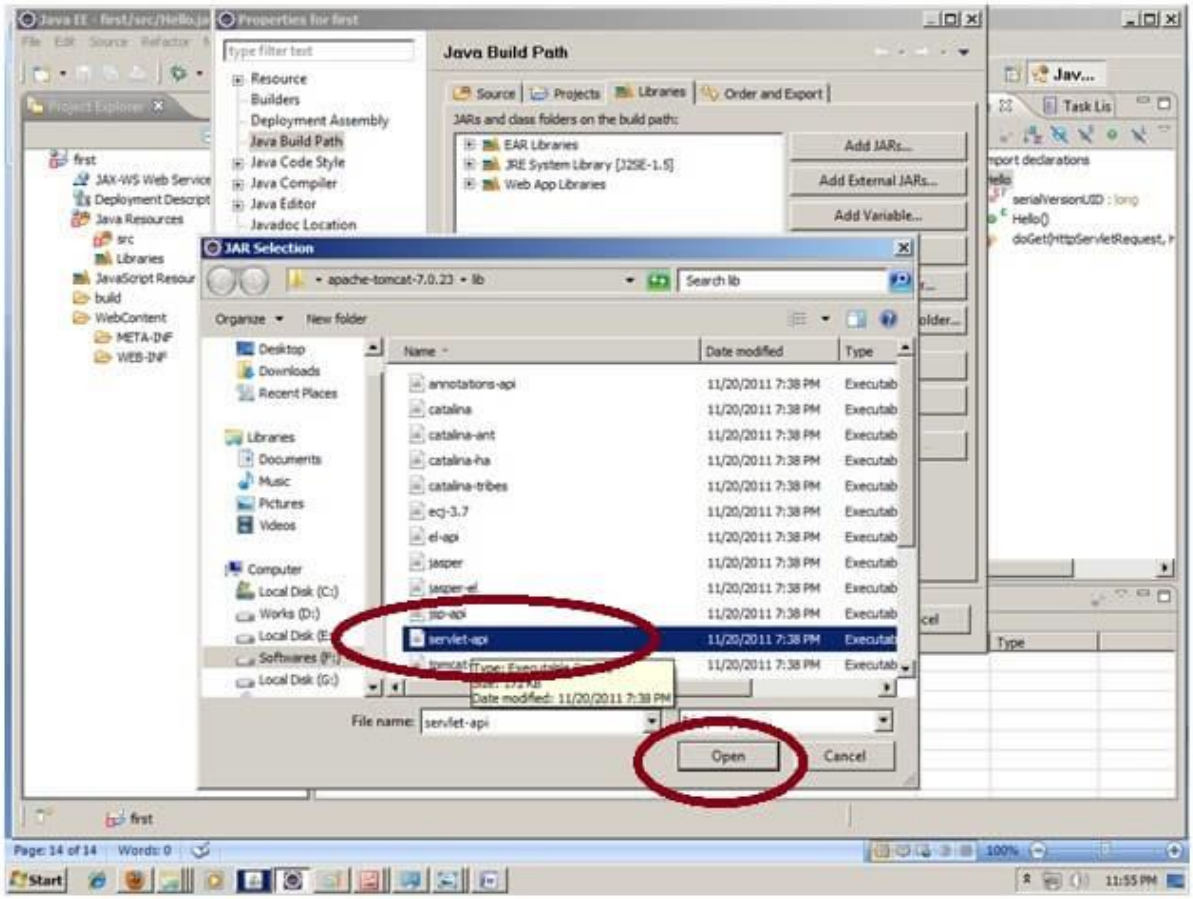
---

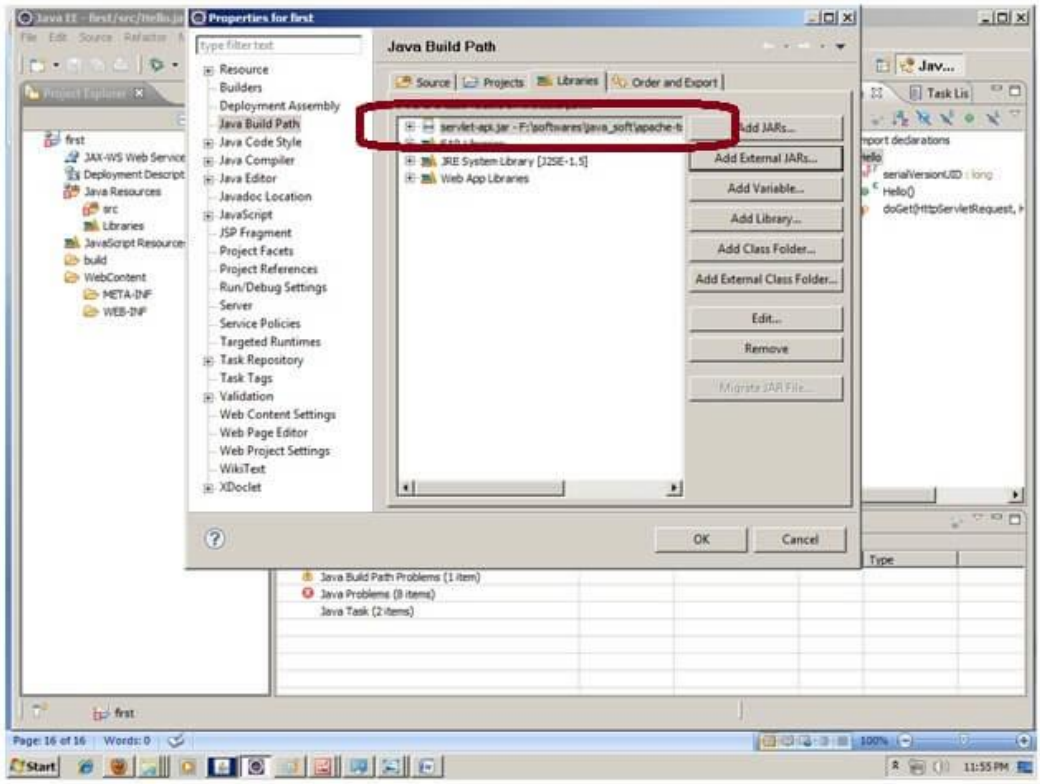
3) add jar file in eclipse IDE:

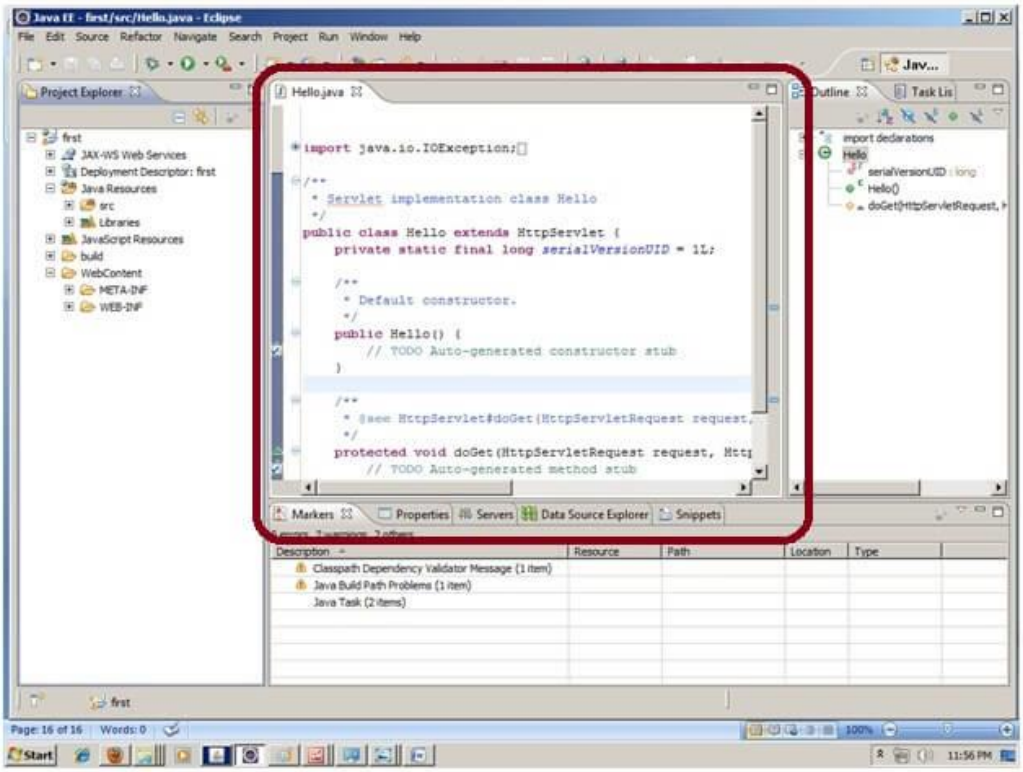
For adding a jar file, **right click on your project -> Build Path -> Configure Build Path -> click on Libraries tab in Java Build Path -> click on Add External JARs button -> select the servlet-api.jar file under tomcat/lib -> ok.**


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
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
Resource
Path
Location
Type
Path Problems (1 item)
Java Problems (8 items)
Java Task (2 items)
first

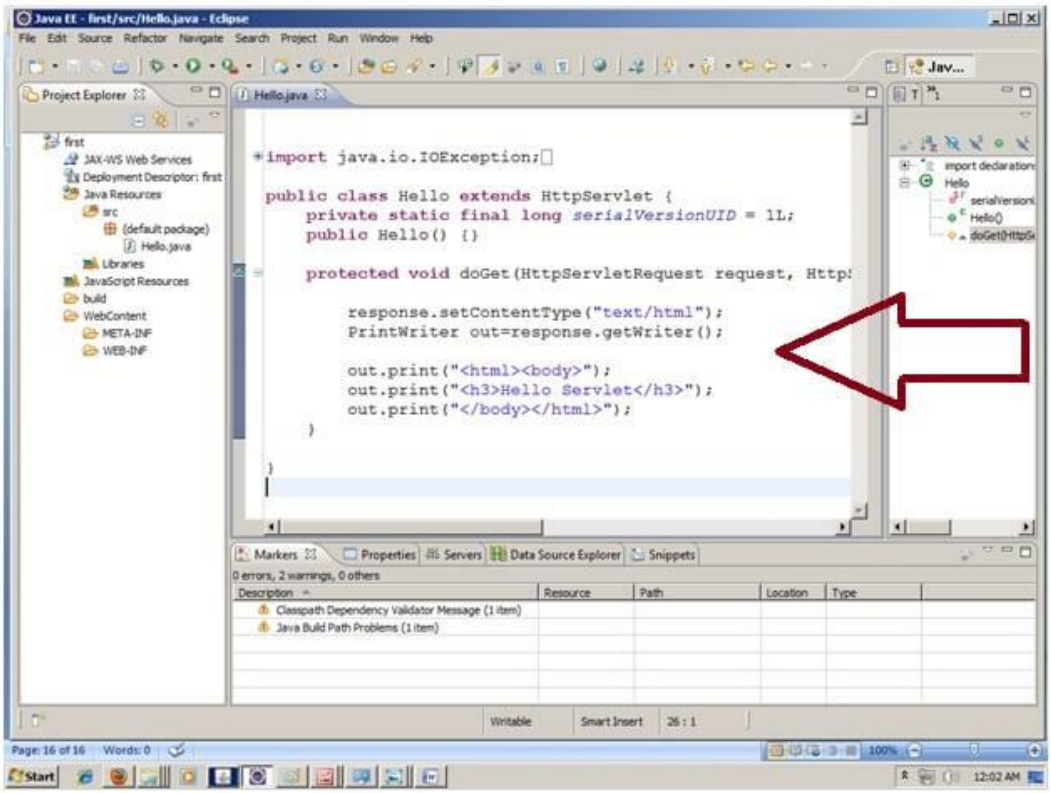

Properties for first
Java Build Path
Resource
Builders
Deployment Assembly
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
JavaScript
JSP Fragment
Project Facets
Project References
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
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


Java Build Path
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...
JAR Selection
apache-tomcat-7.0.23 ▸ lib
Search lib
Organize
New folder
Name
Date modified
Type
annotations-api
catalina
catalina-ant
catalina-ha
catalina-tribes
ecj-3.7
el-api
jasper
jasper-el
jsp-api
servlet-api
11/20/2011 7:38 PM
Executab
File name: servlet-api
Open
Cancel


Java Build Path
Source
Projects
Libraries
Order and Export
servlet-api.jar - F:\softwares\java_soft\apache-t
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)

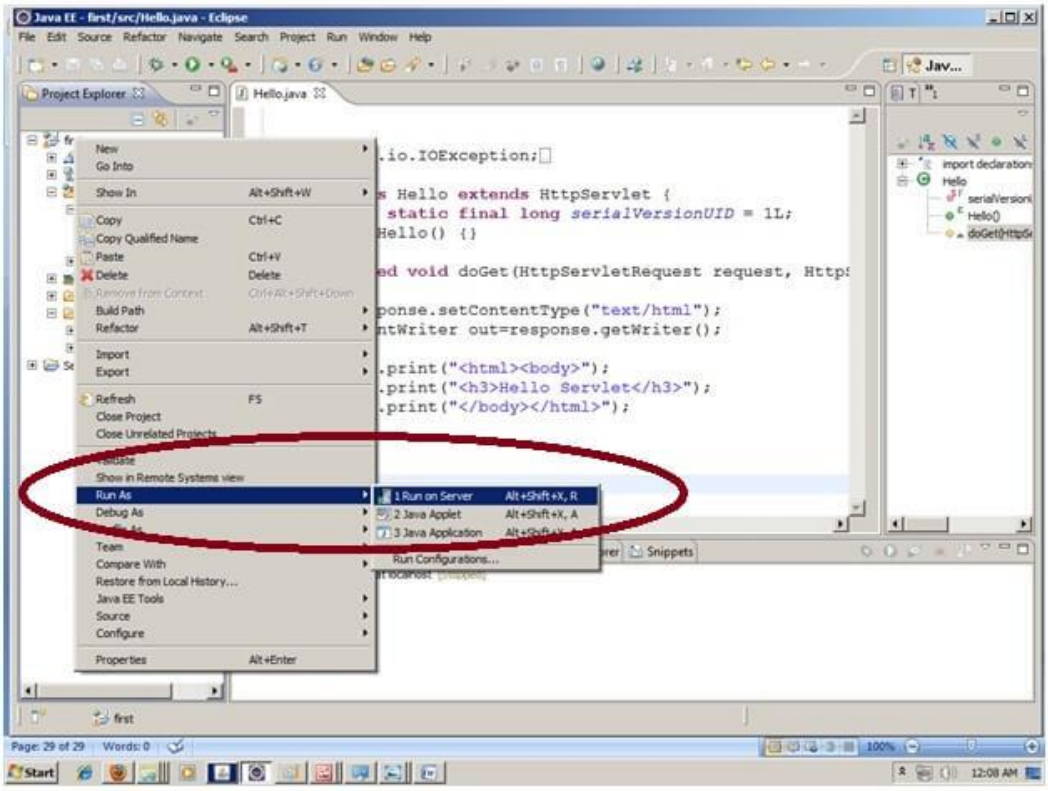


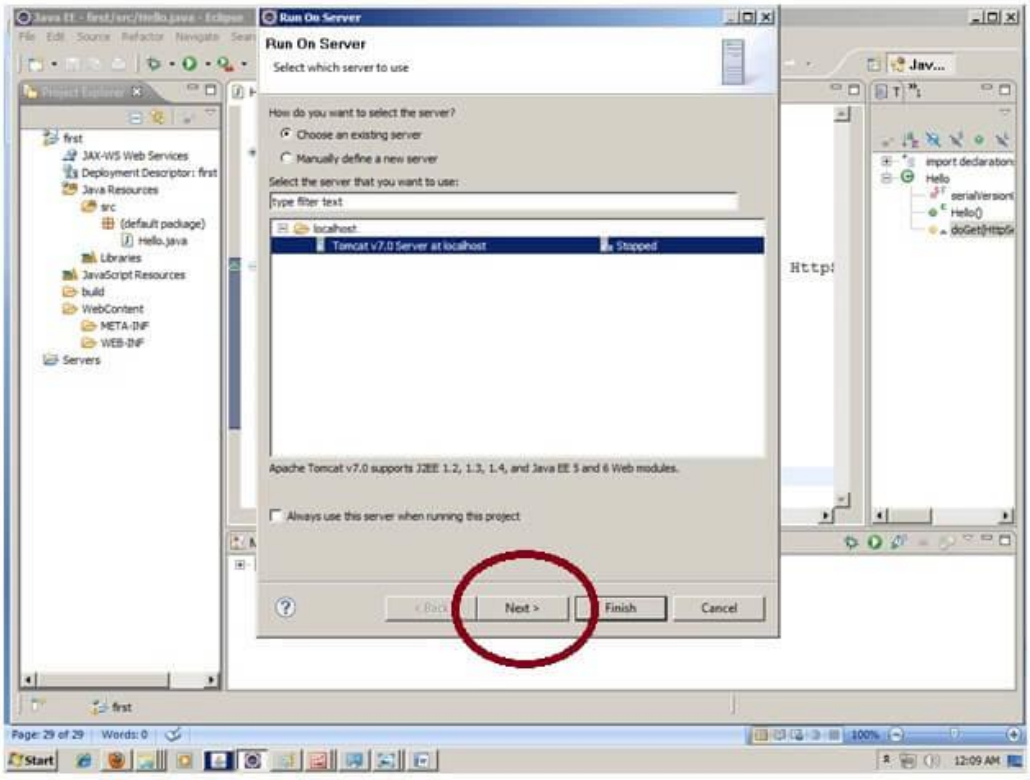
Now servlet has been created, Let's write the first servlet code.

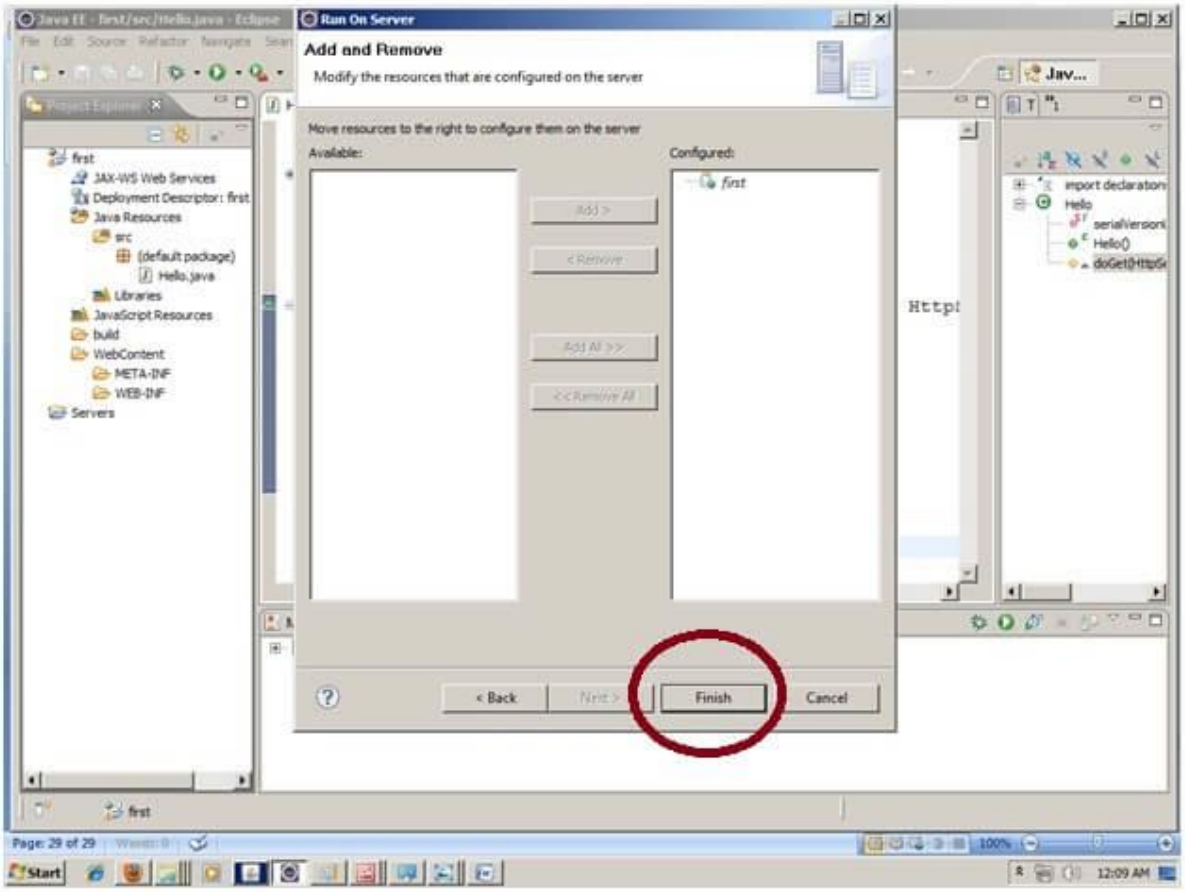


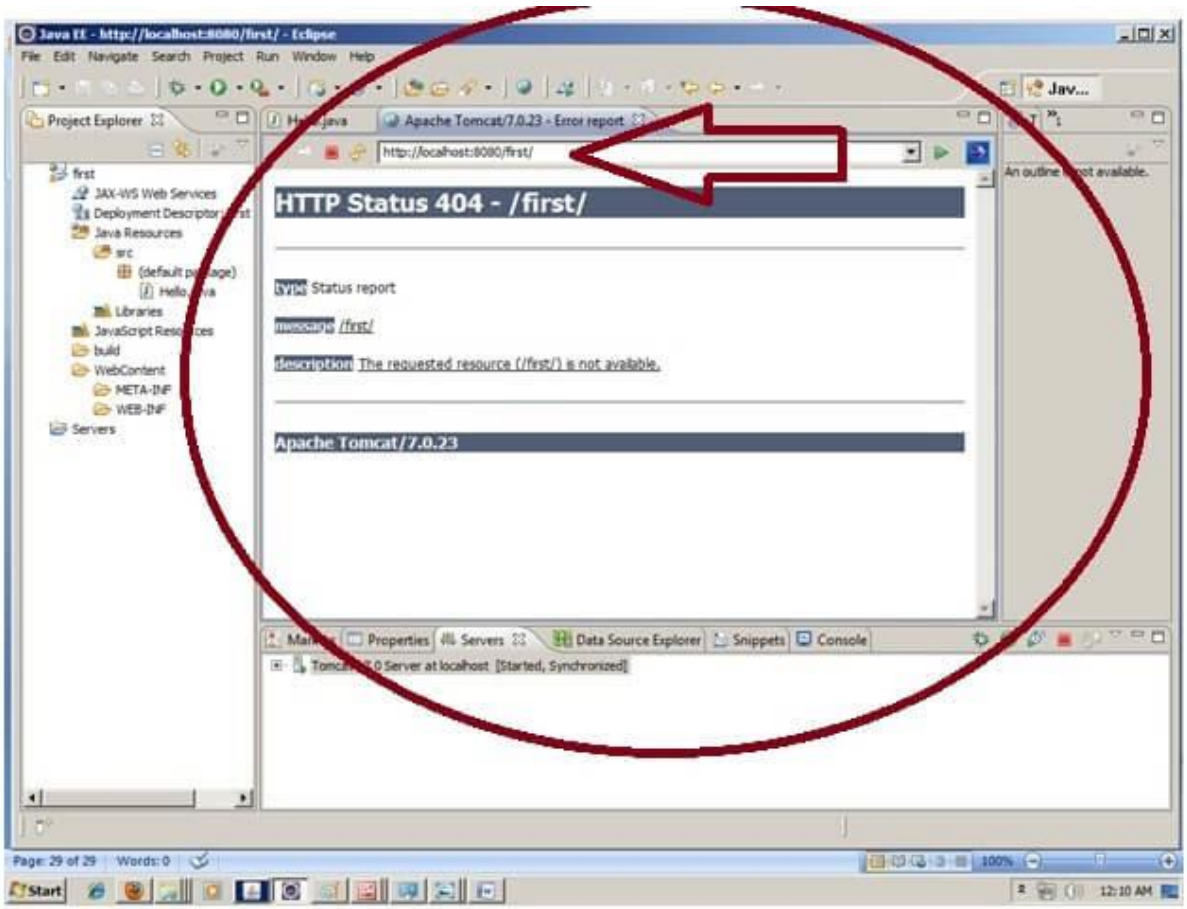
4) Start the server and deploy the project:

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.**

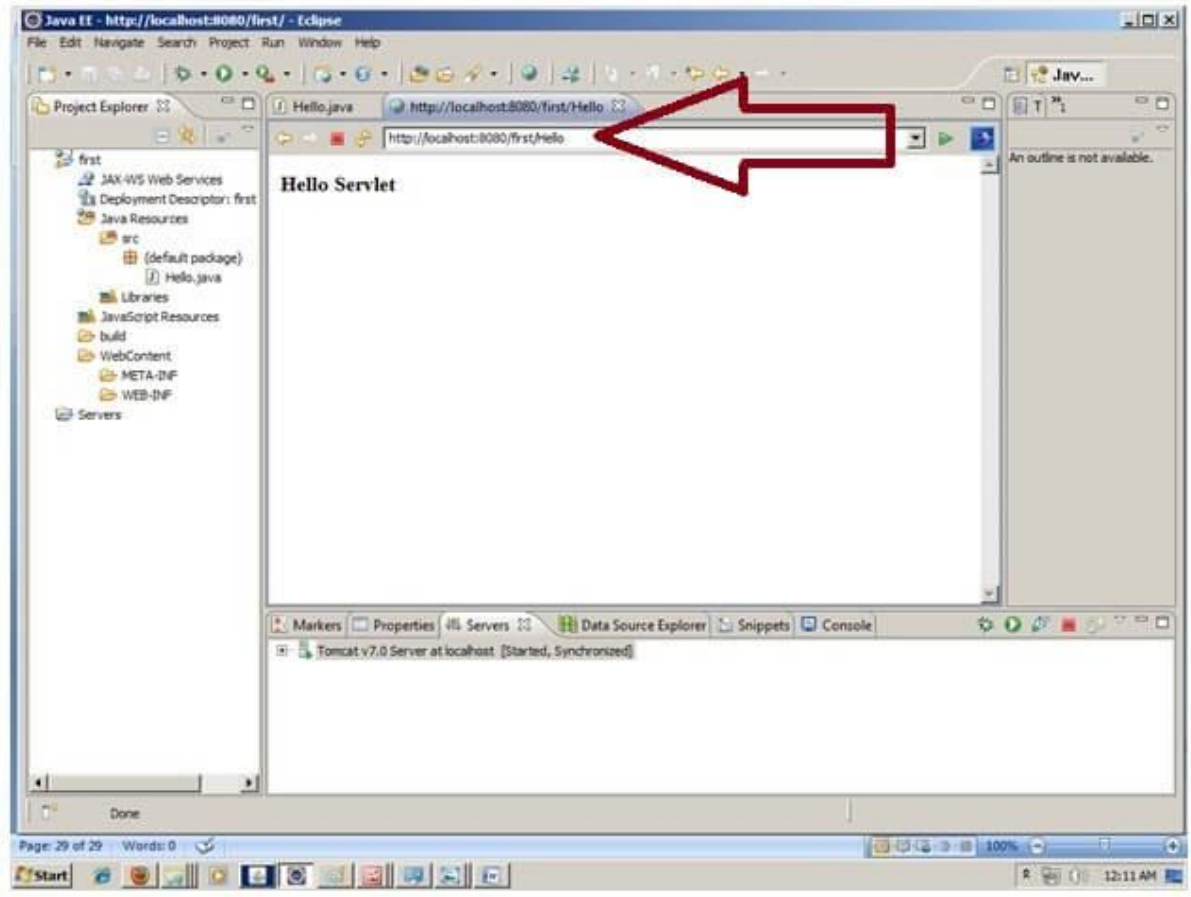





Run On Server
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
Configured:
first
Add >
< Remove
Add All >>
<< Remove All
< Back
Next >
Finish
Cancel


Java EE - http://localhost:8080/first/ - Eclipse
Apache Tomcat/7.0.23 - Error report
http://localhost:8080/first/
HTTP Status 404 - /first/
type Status report
message /first/
description The requested resource (/first/) is not available.
Apache Tomcat/7.0.23
Properties
Servers
Data Source Explorer
Snippets
Console

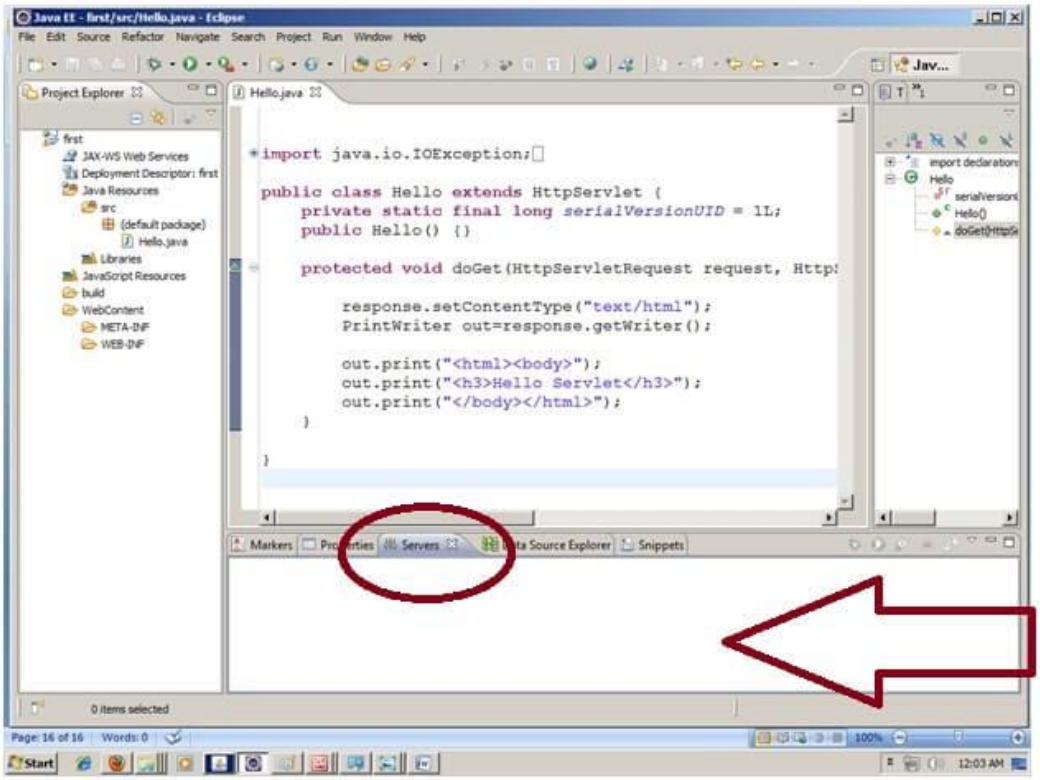
Now tomcat server has been started and project is deployed. To access the servlet write the url pattern name in the URL bar of the browser. In this case Hello then enter.

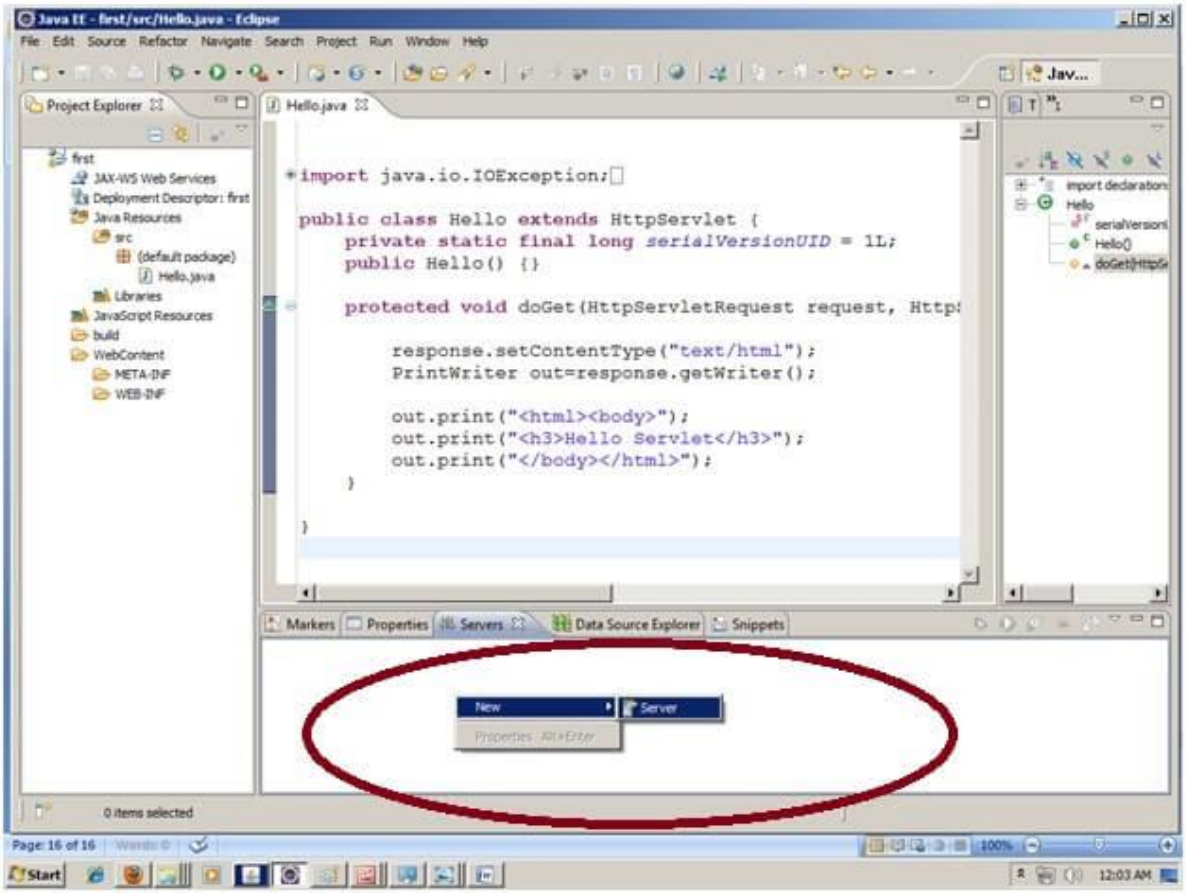


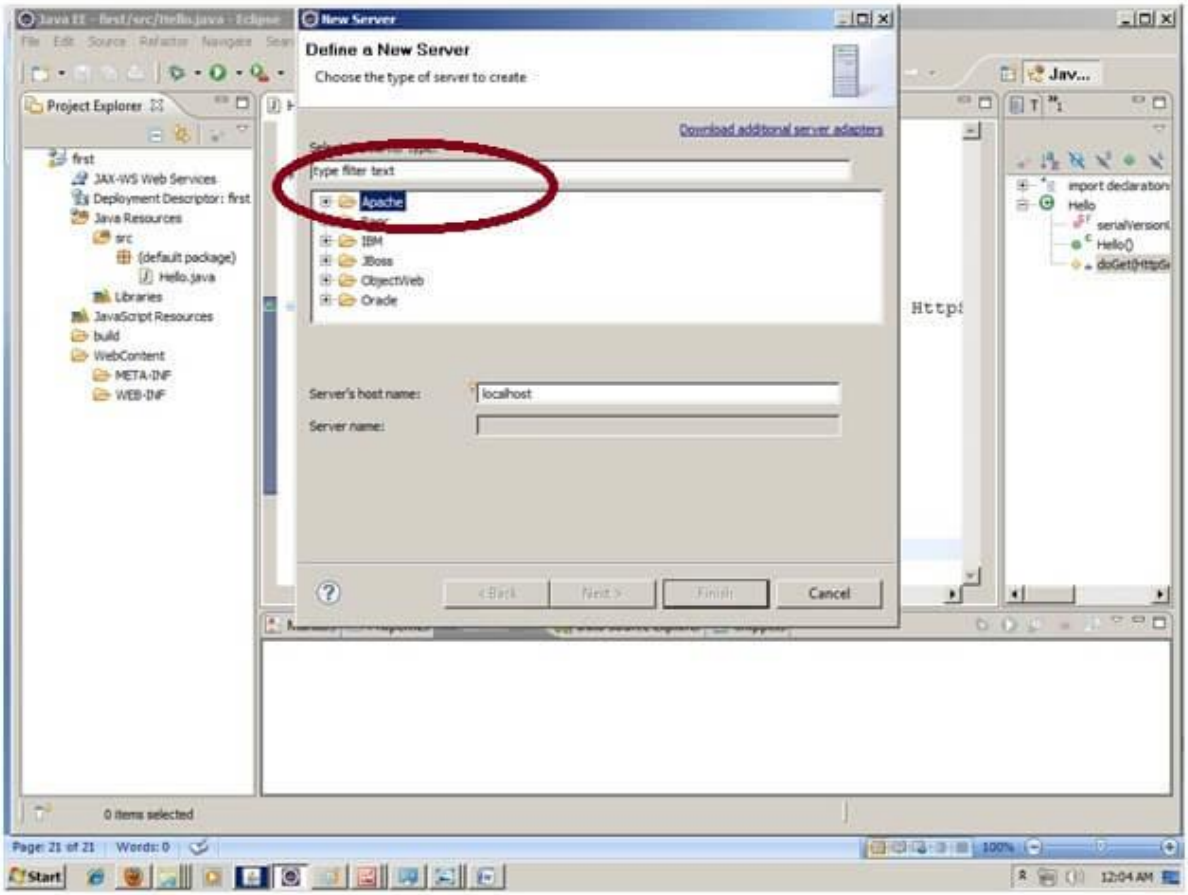
How to configure tomcat server in Eclipse ? (One time Requirement)

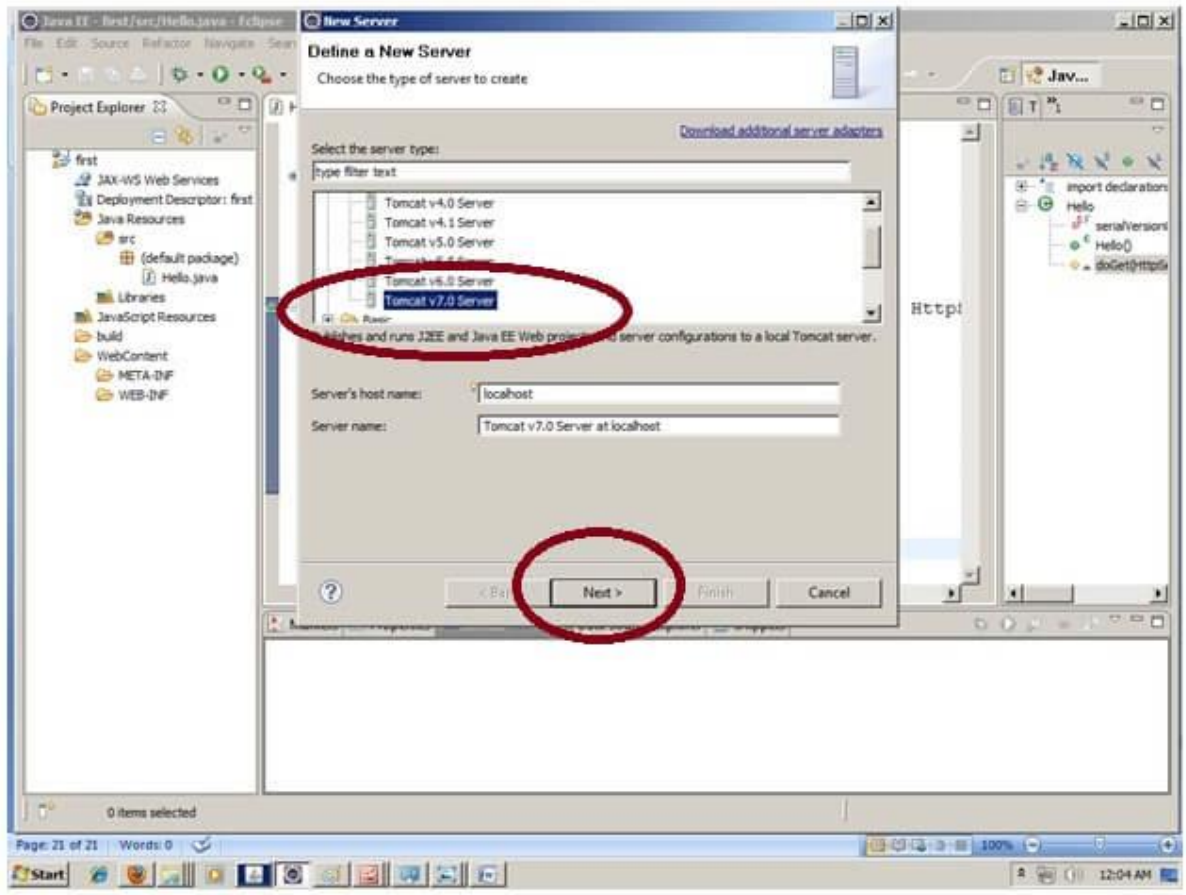
If you are using **Eclipse IDE first time**, you need to configure the tomcat server First.

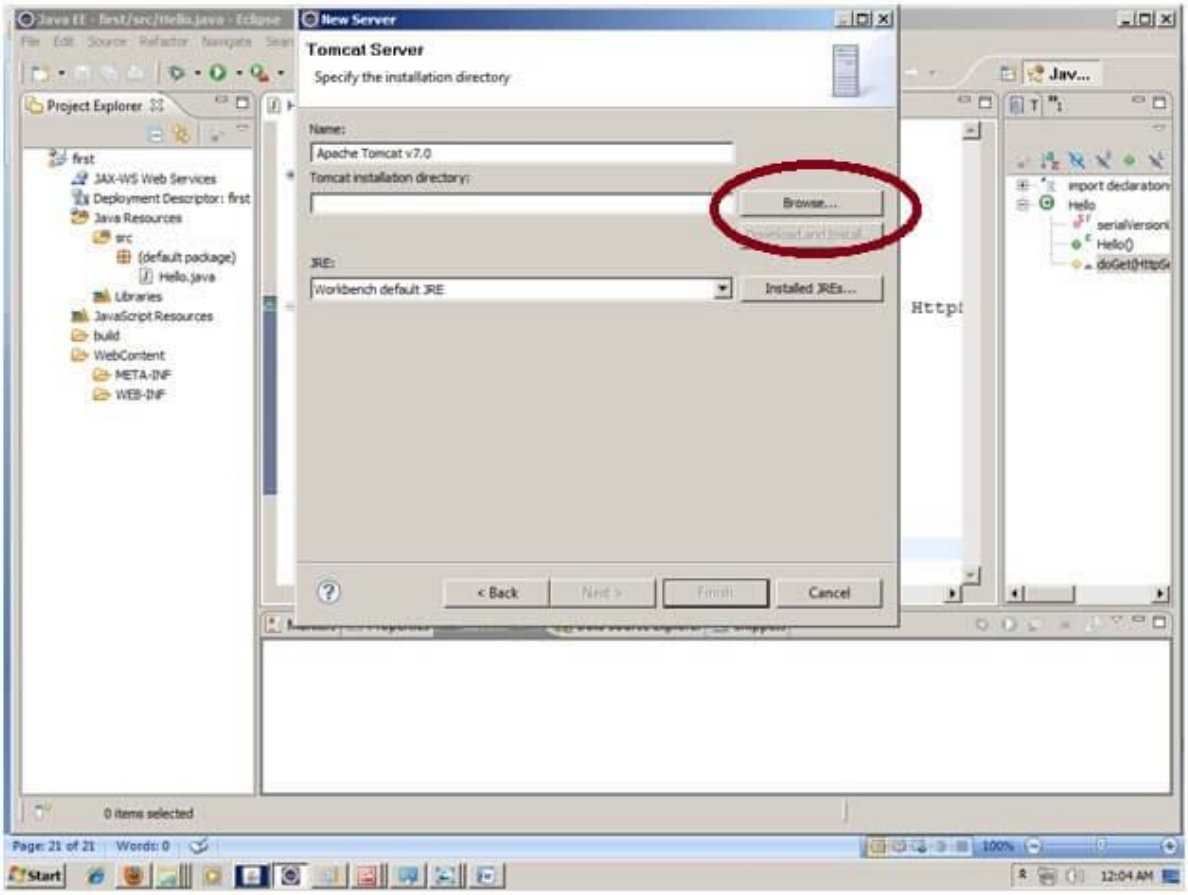
For configuring the tomcat server in eclipse IDE, **click on servers tab at the bottom side of the IDE -> right click on blank area -> New -> Servers -> choose tomcat then its version -> next -> click on Browse button -> select the apache tomcat root folder previous to bin -> next -> addAll -> Finish.**

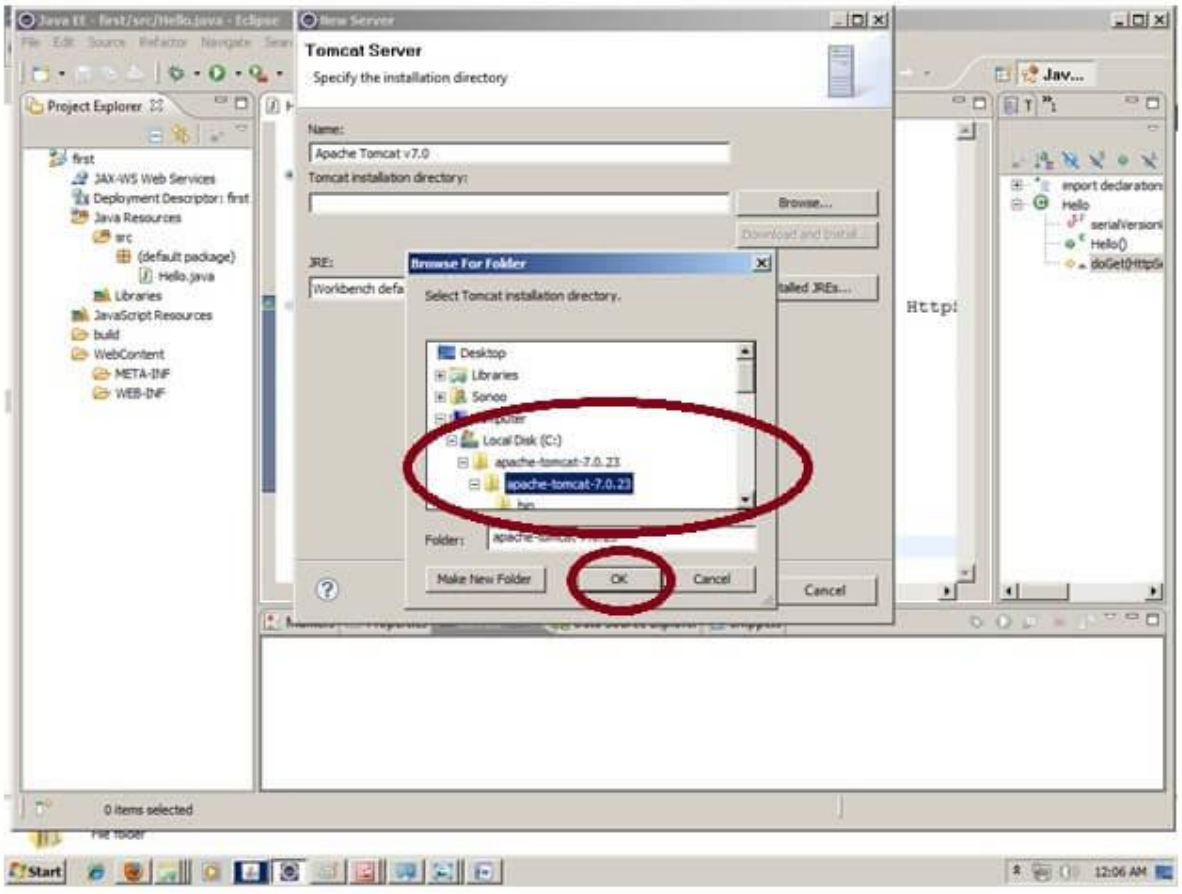

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
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

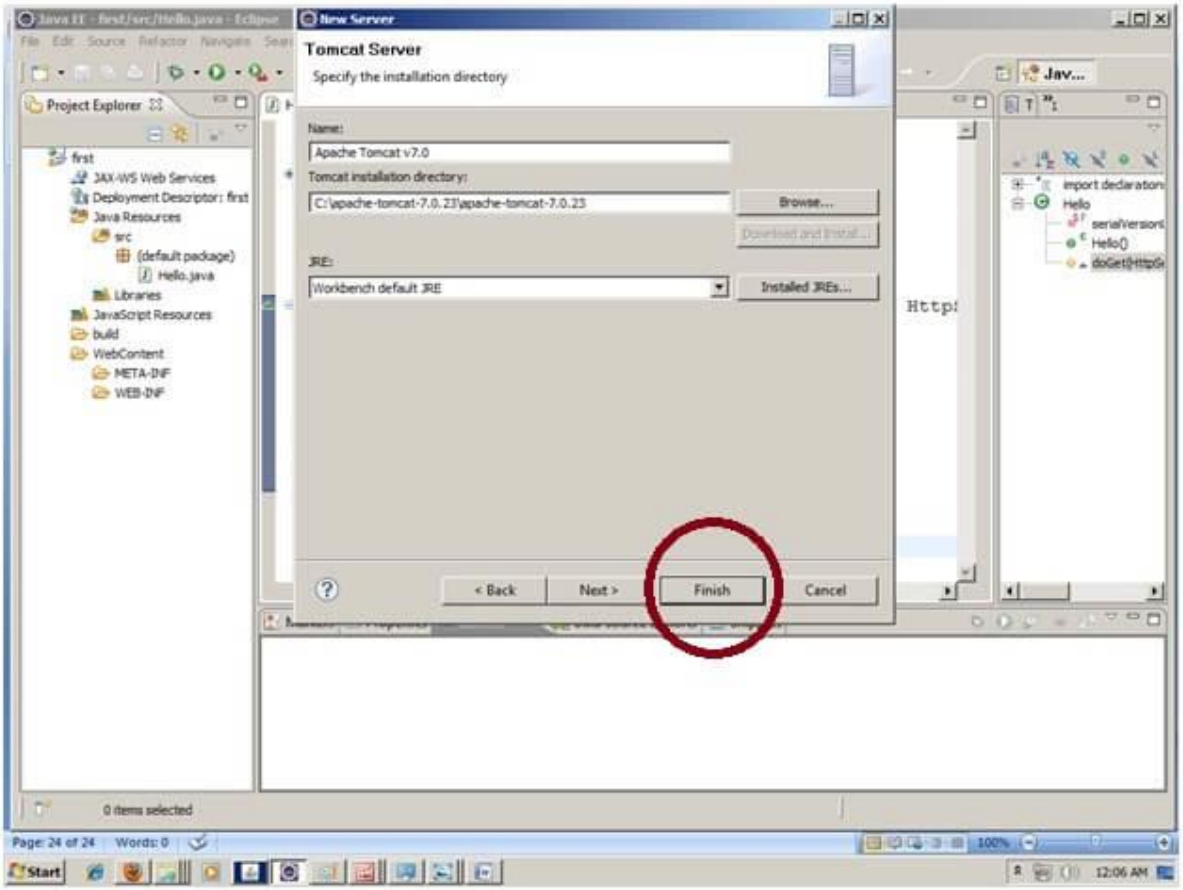

Java EE - first/src/Hello.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Hello.java
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
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
Properties
Servers
Data Source Explorer
Snippets
New
Server
Properties Alt+Enter
0 items selected


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
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Page: 21 of 21
Words: 0
Start
12:04 AM

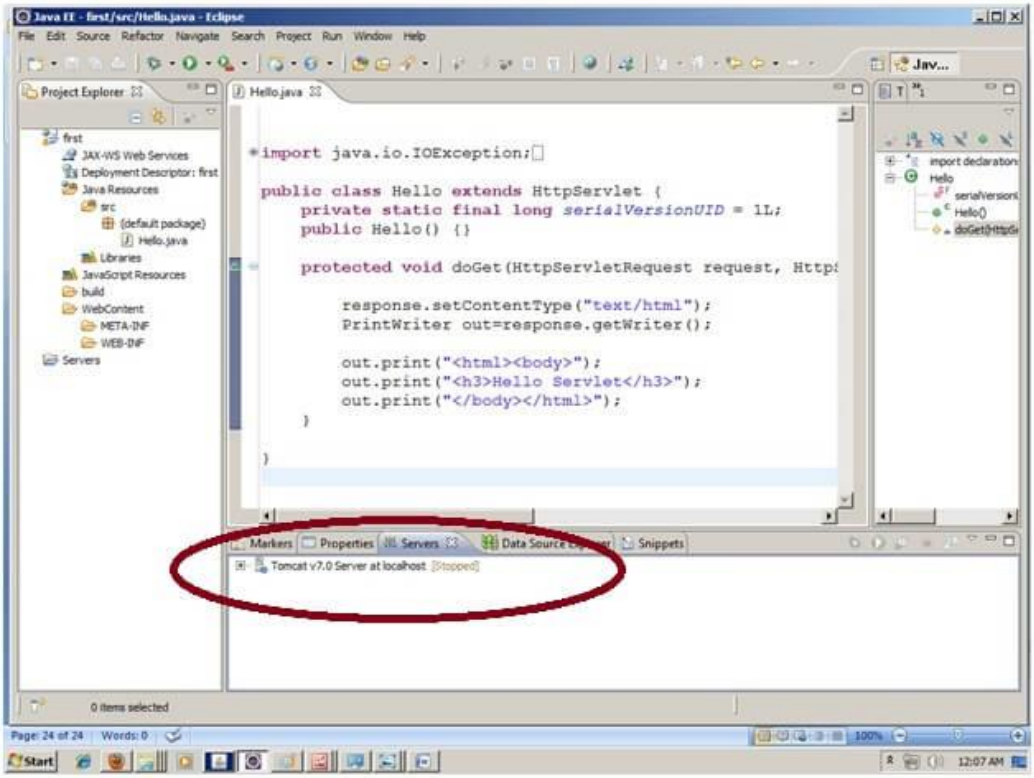

New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Tomcat v4.0 Server
Tomcat v4.1 Server
Tomcat v5.0 Server
Tomcat v6.0 Server
Tomcat v7.0 Server
Server's host name:
localhost
Server name:
Tomcat v7.0 Server at localhost
Next >
Finish
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected
Page: 21 of 21
Words: 0
Start
12:04 AM


New Server
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
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected


Tomcat Server
Specify the installation directory
Name:
Apache Tomcat v7.0
Tomcat installation directory:
Browse...
JRE:
Browse For Folder
Select Tomcat installation directory.
Desktop
Libraries
Local Disk (C:)
apache-tomcat-7.0.23
apache-tomcat-7.0.23
bin
Folder:
Make New Folder
OK
Cancel
Cancel
Project Explorer
first
JAX-WS Web Services
Deployment Descriptor: first
Java Resources
src
(default package)
Hello.java
Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
0 items selected



Now tomcat7 server has been configured in eclipse IDE.



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.

index.html

```
<form action="DemoServ" 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

1. RequestDispatcher Interface
2. Methods of RequestDispatcher interface

1. forward method

2. include method
3. How to get the object of RequestDispatcher
4. Example of RequestDispatcher interface

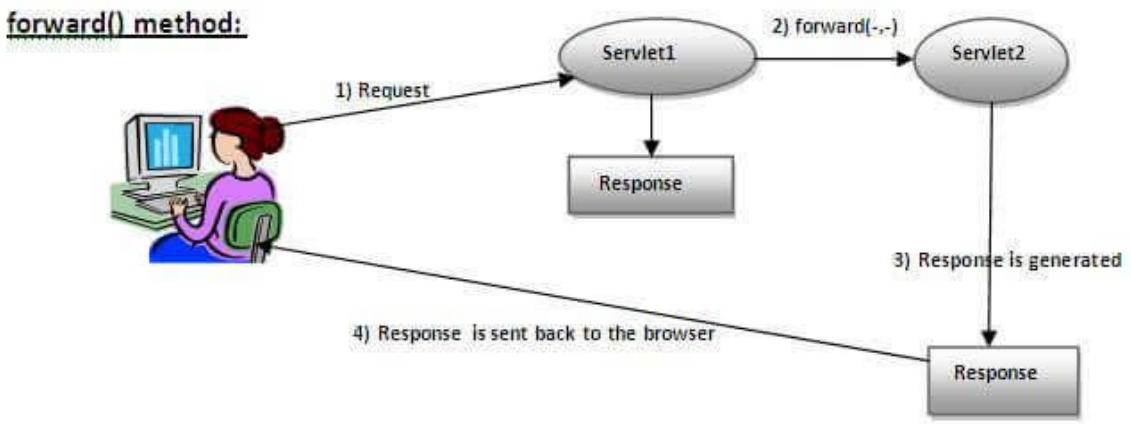
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

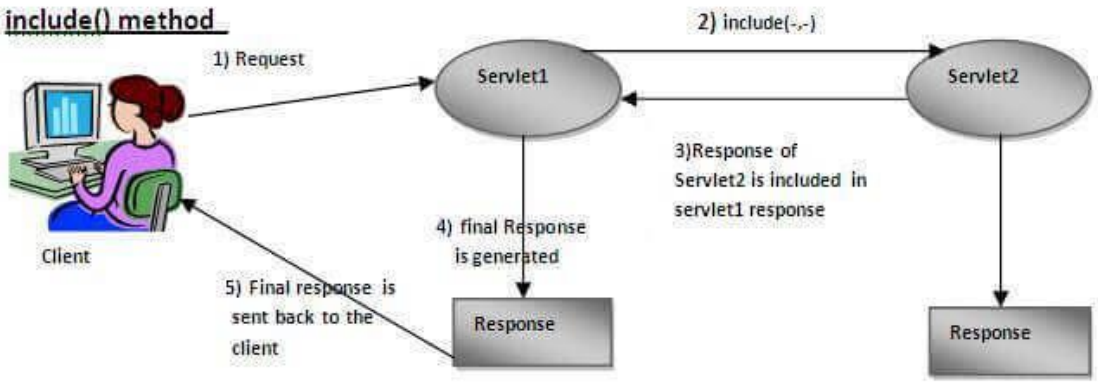
Methods of RequestDispatcher interface

The RequestDispatcher interface provides two methods. They are:

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



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

1. **public** RequestDispatcher getRequestDispatcher(String resource);

Example of using getRequestDispatcher method

1. RequestDispatcher rd=request.getRequestDispatcher("servlet2");
2. //servlet2 is the url-pattern of the second servlet
3.
4. rd.forward(request, response);//method may be include or forward

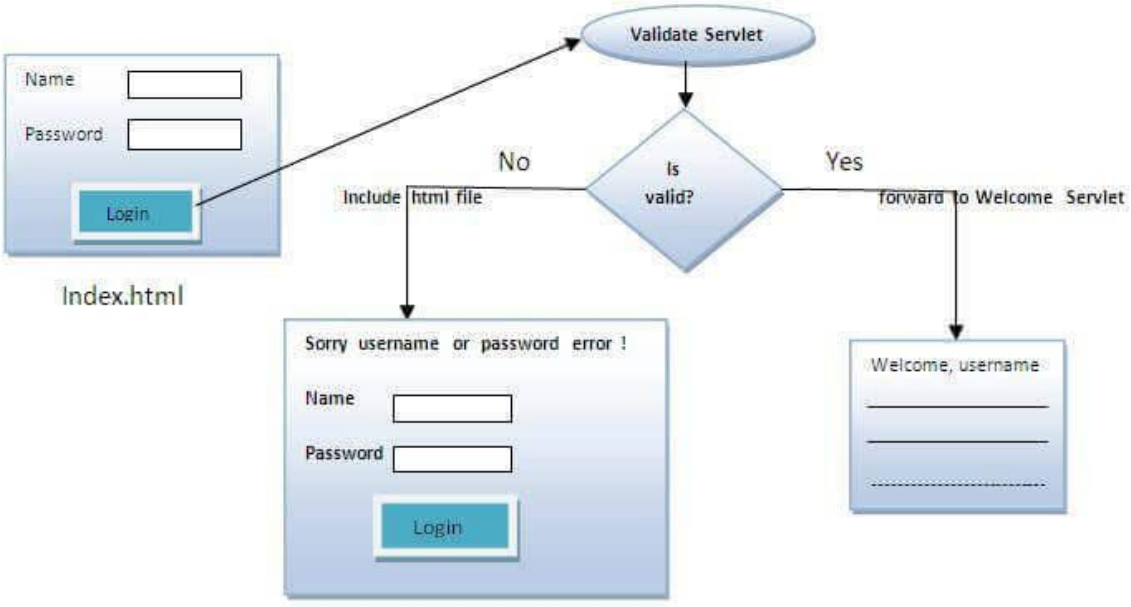
---

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>
```

SendRedirect in servlet

1. sendRedirect method

Syntax of sendRedirect() method

Example of RequestDispatcher interface

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

It accepts relative as well as absolute URL.

It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

Difference between forward() and sendRedirect() method

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

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

Syntax of sendRedirect() method

1. **public void** sendRedirect(String URL)**throws** IOException;

Example of sendRedirect() method

1. response.sendRedirect("http://www.java.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 respons
e)
         throws ServletException, IOException {

      String name=request.getParameter("name");
      response.sendRedirect("https://www.google.co.in/#q="+name);
   }
}
```

ServletConfig Interface

1. ServletConfig Interface

2. Methods of ServletConfig interface
3. How to get the object of ServletConfig
4. Syntax to provide the initialization parameter for a servlet
5. Example of ServletConfig to get initialization parameter
6. Example of ServletConfig to get all the initialization parameter

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.

ServletContext Interface

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

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

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

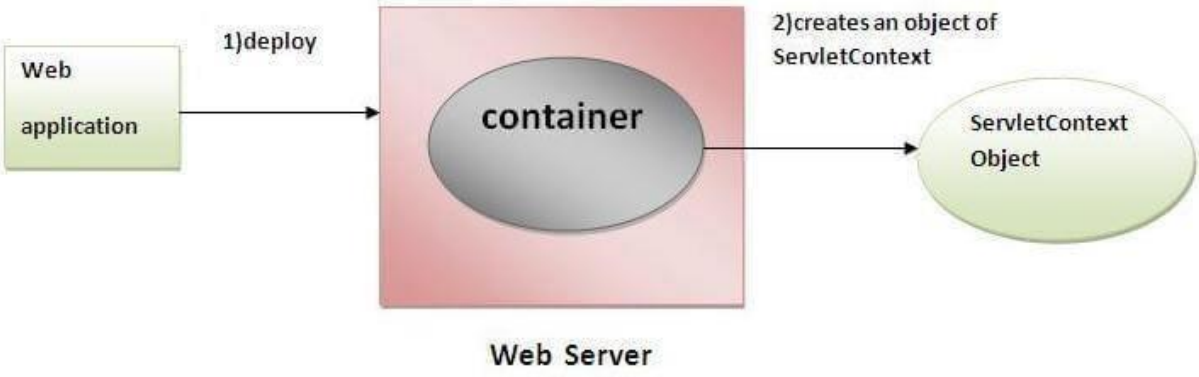
Advantage of ServletContext

Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

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

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



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

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

5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

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

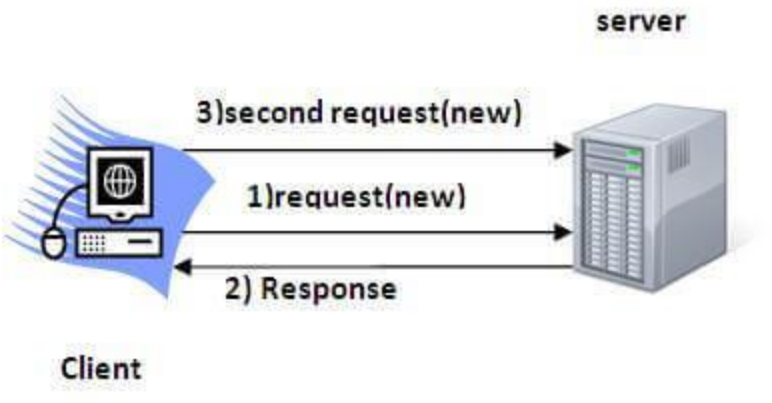
1. Session Tracking
2. Session Tracking Techniques

Session simply means a particular interval of time.

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

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

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



Why use Session Tracking?

To recognize the user

It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

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

Cookies in Servlet

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

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

How Cookie works

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

Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

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

Persistent cookie

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

Advantage of Cookies

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

Disadvantage of Cookies

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

Cookie class

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

Constructor of Cookie class

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

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 b
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 in

1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add
2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all

How to create Cookie?

Let's see the simple code to create cookie.

```
1. Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
2. 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
```

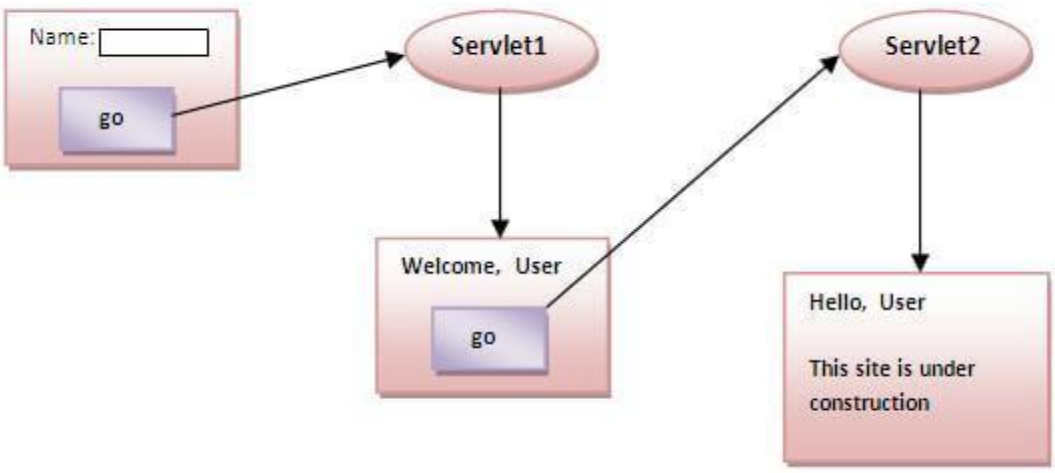
How to get Cookies?

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>
```

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 application, we have created following files.

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

index.html

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

```
<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

```
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 respons
e)
                        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

```
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 respons
e)
                    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

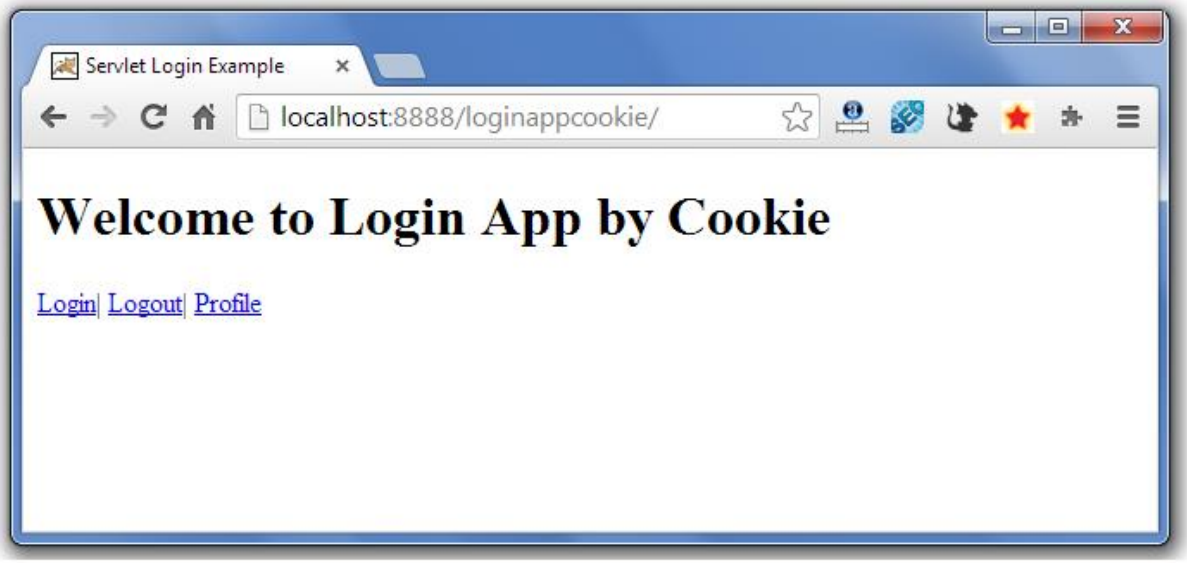
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
                    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();
    }
}
```

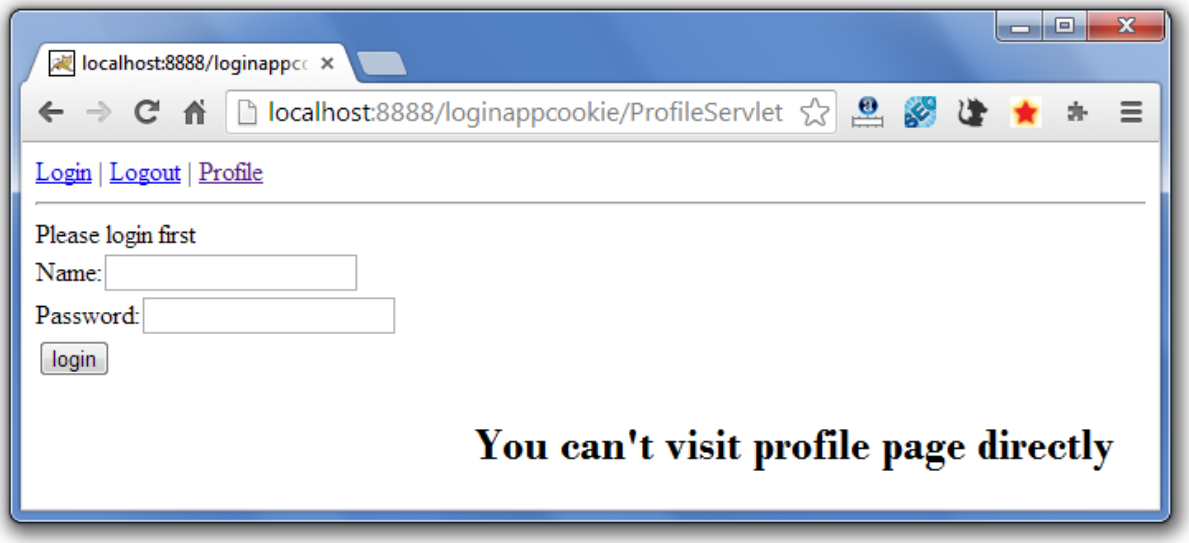
File: web.xml

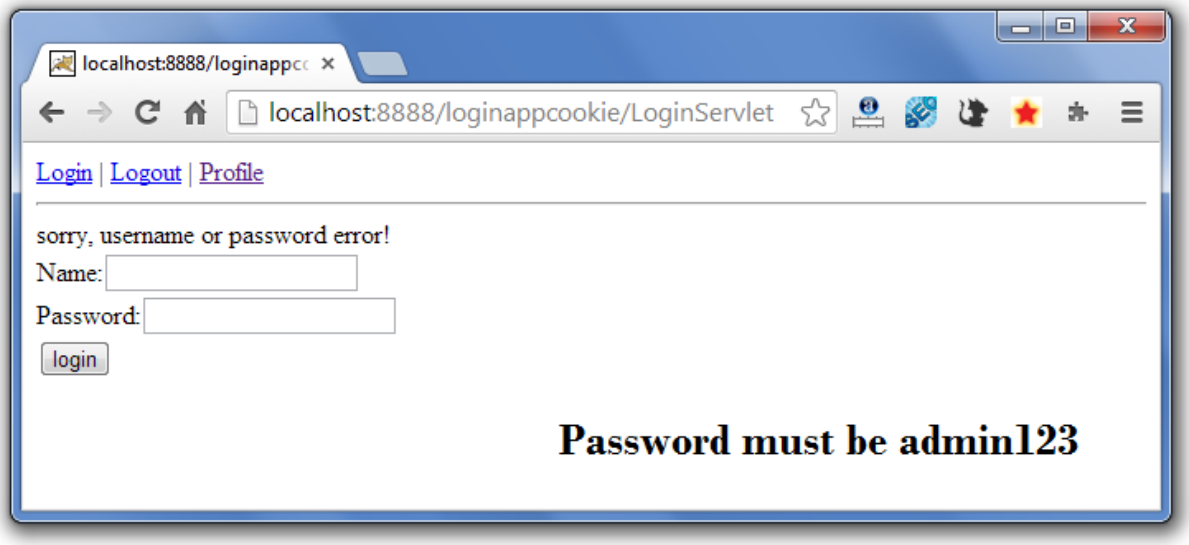
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

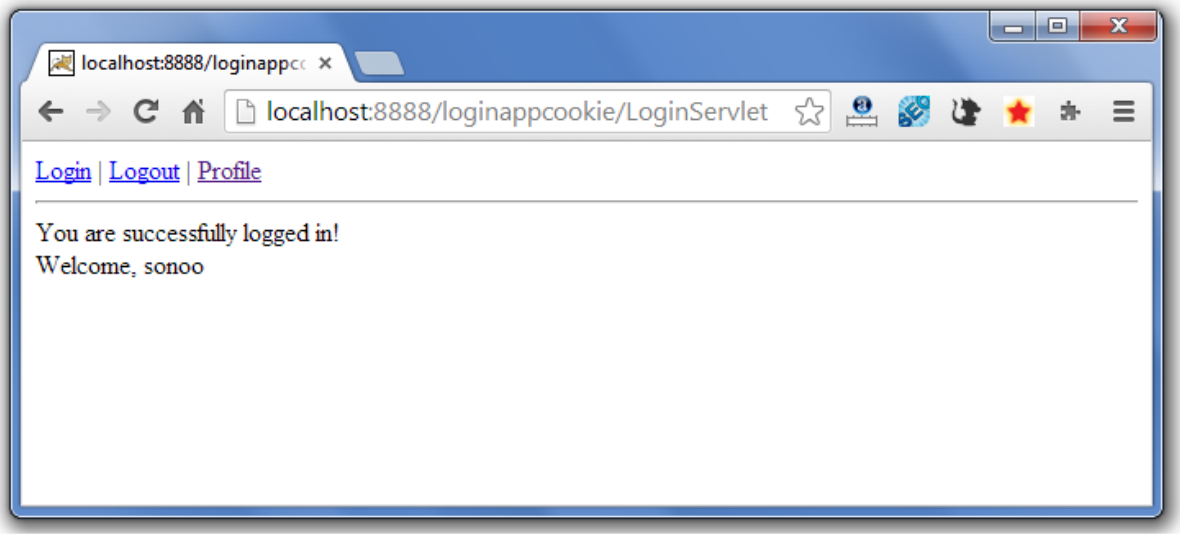
```
xmlns="http://java.sun.com/xml/ns/javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

  <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>
```

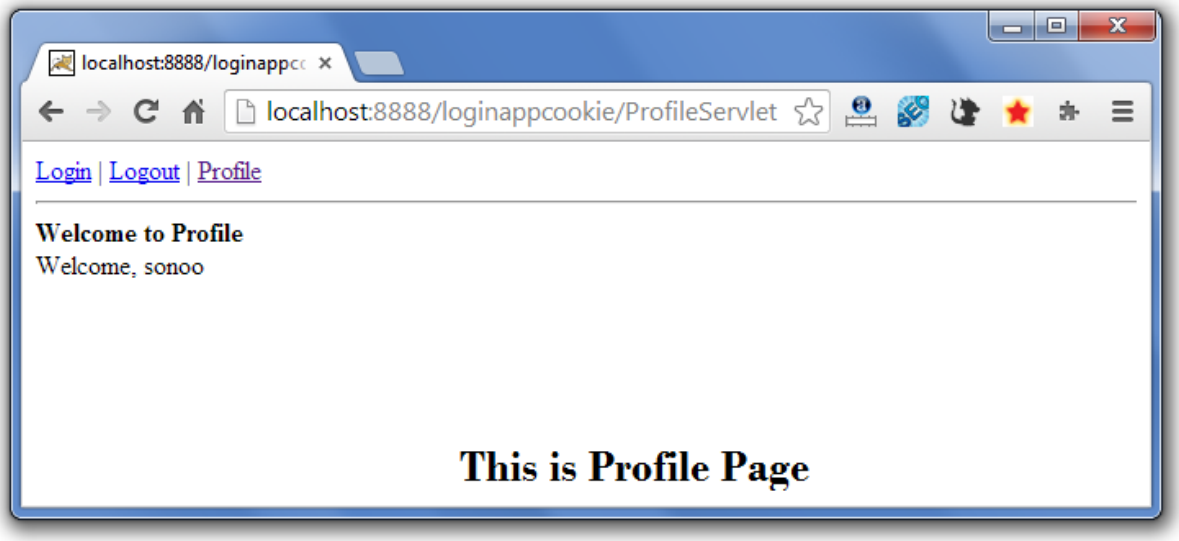
Output

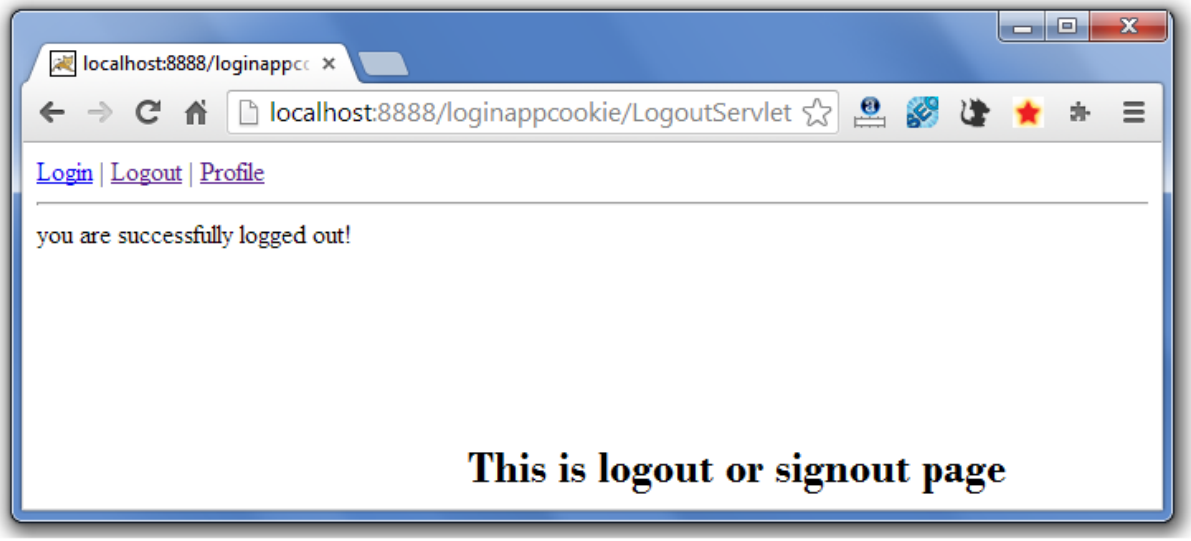

Servlet Login Example
localhost:8888/loginappcookie/
Welcome to Login App by Cookie
Login| Logout| Profile


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


localhost:8888/loginappc
localhost:8888/loginappcookie/ProfileServlet
Login | Logout | Profile
Welcome to Profile
Welcome, sonoo
This is Profile Page


localhost:8888/loginappc
localhost:8888/loginappcookie/LogoutServlet
Login | Logout | Profile
you are successfully logged out!
This is logout or signout page

2) Hidden Form Field

1. Hidden Form Field
2. Example of Hidden Form Field

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

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

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

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

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

Real application of hidden form field

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

Advantage of Hidden Form Field

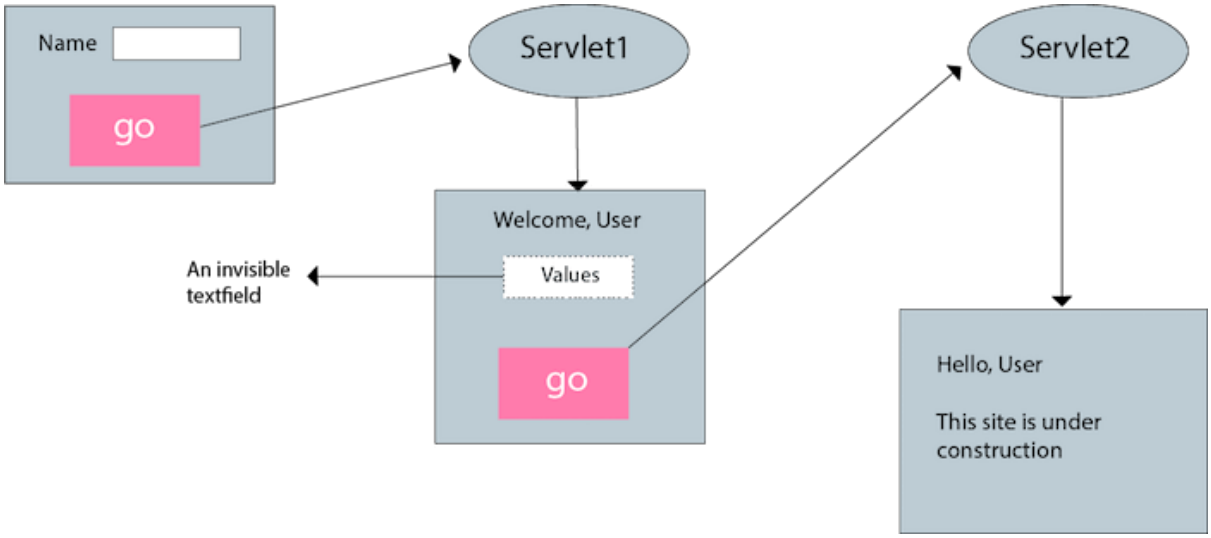
1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

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

Example of using Hidden Form Field

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



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);}
   }
}
```

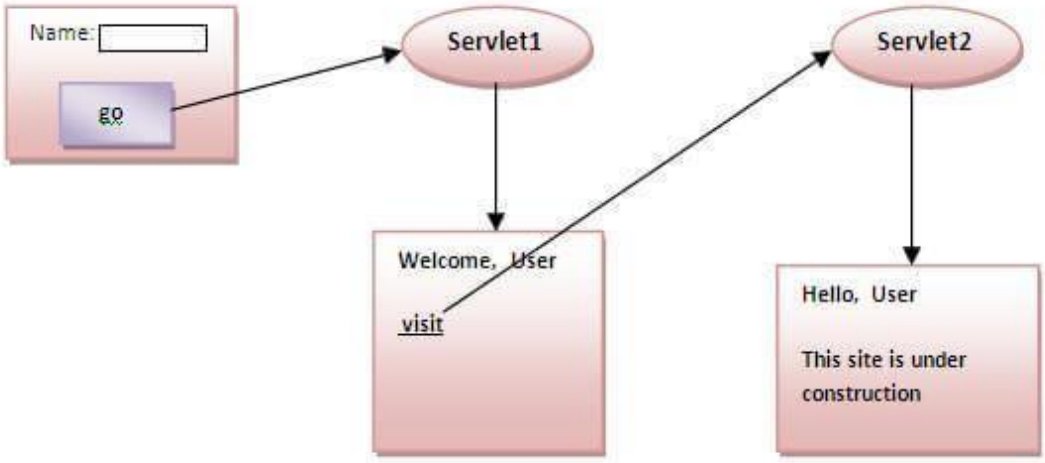
3)URL Rewriting

1. URL Rewriting
2. Advantage of URL Rewriting
3. Disadvantage of URL Rewriting
4. Example of URL Rewriting

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

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

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



Advantage of URL Rewriting

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

Disadvantage of URL Rewriting

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

Example of using URL Rewriting

In this example, we are 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);}
    }


}
```

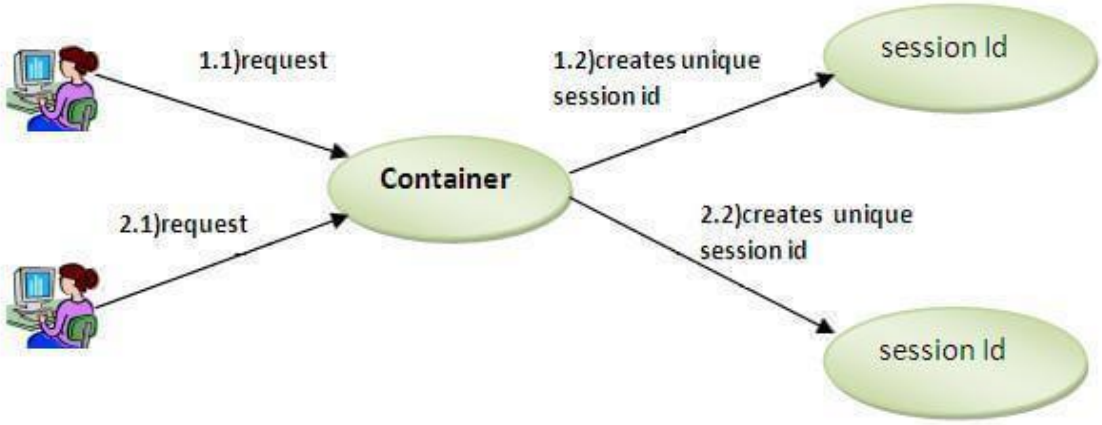
4) HttpSession interface

1. HttpSession interface

2. How to get the HttpSession object
3. Commonly used methods of HttpSession interface
4. Example of using HttpSession

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

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



How to get the HttpSession object ?

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

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

Commonly used methods of HttpSession interface

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

Example of using HttpSession

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

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);}
    }


}
```

Servlet CRUD example

Create "user905" table in Oracle Database with auto incrementing id using sequence. There are 5 fields in it: id, name, password, email and country.

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER	No	-	1
NAME	VARCHAR2(4000)	Yes	-	-
PASSWORD	VARCHAR2(4000)	Yes	-	-
EMAIL	VARCHAR2(4000)	Yes	-	-
COUNTRY	VARCHAR2(4000)	Yes	-	-
				1 - 5

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
```

```
<h1>Add New Employee</h1>
<form action="SaveServlet" method="post">
<table>
<tr><td>Name:</td><td><input type="text" name="name"/></td></tr>
<tr><td>Password:</td><td><input type="password" name="password"/></td><
/tr>
<tr><td>Email:</td><td><input type="email" name="email"/></td></tr>
<tr><td>Country:</td><td>
<select name="country" style="width:150px">
<option>India</option>
<option>USA</option>
<option>UK</option>
<option>Other</option>
</select>
</td></tr>
<tr><td colspan="2"><input type="submit" value="Save Employee"/></td></tr>
</table>
</form>

<br/>
<a href="ViewServlet">view employees</a>

</body>
</html>
```

File: Emp.java

```
public class Emp {
private int id;
private String name,password,email,country;
public int getId() {
   return id;
}
public void setId(int id) {
   this.id = id;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}
public String getPassword() {
   return password;
```

```
}
public void setPassword(String password) {
   this.password = password;
}
public String getEmail() {
   return email;
}
public void setEmail(String email) {
   this.email = email;
}
public String getCountry() {
   return country;
}
public void setCountry(String country) {
   this.country = country;
}

}
```

File: EmpDao.java

```
import java.util.*;
import java.sql.*;

public class EmpDao {

   public static Connection getConnection(){
      Connection con=null;
      try{
         Class.forName("oracle.jdbc.driver.OracleDriver");
         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","s
ystem","oracle");
      }catch(Exception e){System.out.println(e);}
      return con;
   }
   public static int save(Emp e){
      int status=0;
      try{
         Connection con=EmpDao.getConnection();
         PreparedStatement ps=con.prepareStatement(
                  "insert into user905(name,password,email,country) values (?,?,?,?)
");
         ps.setString(1,e.getName());
         ps.setString(2,e.getPassword());
         ps.setString(3,e.getEmail());
```

```
            ps.setString(4,e.getCountry());

            status=ps.executeUpdate();

            con.close();
        }catch(Exception ex){ex.printStackTrace();}

        return status;
    }
    public static int update(Emp e){
        int status=0;
        try{
            Connection con=EmpDao.getConnection();
            PreparedStatement ps=con.prepareStatement(
                    "update user905 set name=?,password=?,email=?,country=? wher
e id=?");
            ps.setString(1,e.getName());
            ps.setString(2,e.getPassword());
            ps.setString(3,e.getEmail());
            ps.setString(4,e.getCountry());
            ps.setInt(5,e.getId());

            status=ps.executeUpdate();

            con.close();
        }catch(Exception ex){ex.printStackTrace();}

        return status;
    }
    public static int delete(int id){
        int status=0;
        try{
            Connection con=EmpDao.getConnection();
            PreparedStatement ps=con.prepareStatement("delete from user905 where i
d=?");
            ps.setInt(1,id);
            status=ps.executeUpdate();

            con.close();
        }catch(Exception e){e.printStackTrace();}

        return status;
    }
    public static Emp getEmployeeById(int id){
```

```
        Emp e=new Emp();

        try{
            Connection con=EmpDao.getConnection();
            PreparedStatement ps=con.prepareStatement("select * from user905 where
id=?");
            ps.setInt(1,id);
            ResultSet rs=ps.executeQuery();
            if(rs.next()){
                e.setId(rs.getInt(1));
                e.setName(rs.getString(2));
                e.setPassword(rs.getString(3));
                e.setEmail(rs.getString(4));
                e.setCountry(rs.getString(5));
            }
            con.close();
        }catch(Exception ex){ex.printStackTrace();}

        return e;
    }
    public static List<Emp> getAllEmployees(){
        List<Emp> list=new ArrayList<Emp>();

        try{
            Connection con=EmpDao.getConnection();
            PreparedStatement ps=con.prepareStatement("select * from user905");
            ResultSet rs=ps.executeQuery();
            while(rs.next()){
                Emp e=new Emp();
                e.setId(rs.getInt(1));
                e.setName(rs.getString(2));
                e.setPassword(rs.getString(3));
                e.setEmail(rs.getString(4));
                e.setCountry(rs.getString(5));
                list.add(e);
            }
            con.close();
        }catch(Exception e){e.printStackTrace();}

        return list;
    }
}
```

File: SaveServlet.java

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

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/SaveServlet")
public class SaveServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse respons
e)
        throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();

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

       Emp e=new Emp();
       e.setName(name);
       e.setPassword(password);
       e.setEmail(email);
       e.setCountry(country);

       int status=EmpDao.save(e);
       if(status>0){
          out.print("<p>Record saved successfully!</p>");
          request.getRequestDispatcher("index.html").include(request, response);
       }else{
          out.println("Sorry! unable to save record");
       }

       out.close();
   }

}
```

File: EditServlet.java

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

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/EditServlet")
public class EditServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
         throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();
       out.println("<h1>Update Employee</h1>");
       String sid=request.getParameter("id");
       int id=Integer.parseInt(sid);

       Emp e=EmpDao.getEmployeeById(id);

       out.print("<form action='EditServlet2' method='post'>");
       out.print("<table>");
       out.print("<tr><td></td><td><input type='hidden' name='id' value='"+e.get
Id()+"'/></td></tr>");
       out.print("<tr><td>Name:</td><td><input type='text' name='name' value='
"+e.getName()+"'/></td></tr>");
       out.print("<tr><td>Password:</td><td><input type='password' name='pass
word' value='"+e.getPassword()+"'/>
             </td></tr>");
       out.print("<tr><td>Email:</td><td><input type='email' name='email' value=
'"+e.getEmail()+"'/></td></tr>");
       out.print("<tr><td>Country:</td><td>");
       out.print("<select name='country' style='width:150px'>");
       out.print("<option>India</option>");
       out.print("<option>USA</option>");
       out.print("<option>UK</option>");
       out.print("<option>Other</option>");
       out.print("</select>");
       out.print("</td></tr>");
       out.print("<tr><td colspan='2'><input type='submit' value='Edit & Save '/></
td></tr>");
       out.print("</table>");
       out.print("</form>");

       out.close();
   }
```

```
}
```
File: EditServlet2.java

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

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/EditServlet2")
public class EditServlet2 extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse respons
e)
         throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();

       String sid=request.getParameter("id");
       int id=Integer.parseInt(sid);
       String name=request.getParameter("name");
       String password=request.getParameter("password");
       String email=request.getParameter("email");
       String country=request.getParameter("country");

       Emp e=new Emp();
       e.setId(id);
       e.setName(name);
       e.setPassword(password);
       e.setEmail(email);
       e.setCountry(country);

       int status=EmpDao.update(e);
       if(status>0){
          response.sendRedirect("ViewServlet");
       }else{
          out.println("Sorry! unable to update record");
       }

       out.close();
   }

}
```
File: DeleteServlet.java

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/DeleteServlet")
public class DeleteServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
          throws ServletException, IOException {
       String sid=request.getParameter("id");
       int id=Integer.parseInt(sid);
       EmpDao.delete(id);
       response.sendRedirect("ViewServlet");
   }
}
```

File: ViewServlet.java

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

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/ViewServlet")
public class ViewServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse respons
e)
            throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();
       out.println("<a href='index.html'>Add New Employee</a>");
       out.println("<h1>Employees List</h1>");

       List<Emp> list=EmpDao.getAllEmployees();

       out.print("<table border='1' width='100%'");
       out.print("<tr><th>Id</th><th>Name</th><th>Password</th><th>Email</
th><th>Country</th>
              <th>Edit</th><th>Delete</th></tr>");
```

```
        for(Emp e:list){
         out.print("<tr><td>"+e.getId()+"</td><td>"+e.getName()+"</td><td>"+e.
getPassword()+"</td>
                  <td>"+e.getEmail()+"</td><td>"+e.getCountry()+"</td><td><a href
='EditServlet?id="+e.getId()+"'>edit</a></td>
                  <td><a href='DeleteServlet?id="+e.getId()+"'>delete</a></td></tr>"
);
        }
        out.print("</table>");

        out.close();
    }
}
```