Java Annotations

Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

First, we will learn some built-in annotations then we will move on creating and using custom annotations.

Built-In Java Annotations

There are several built-in annotations in Java. Some annotations are applied to Java code and some to other annotations.

Built-In Java Annotations used in Java code

- @Override
- @SuppressWarnings
- @Deprecated

Built-In Java Annotations used in other annotations

- @Target
- @Retention
- @Inherited
- @Documented

Understanding Built-In Annotations

Let's understand the built-in annotations first.

@Override

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs.

Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

```
//TestAnnotaion1.java

class Animal{
void eatSomething(){
System.out.println("eating something”);
}
}

class Dog extends Animal{
@Override
void eatsomething(){
System.out.println("eating foods”);
}//should be eatSomething
}

class TestAnnotation1{
public static void main(String args[]){
Animal a=new Dog();
a.eatSomething();
}}
```

@SuppressWarnings

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
import java.util.*;
class TestAnnotation2{
@SuppressWarnings("unchecked")
public static void main(String args[]){
ArrayList list=new ArrayList();
list.add("sonoo");
list.add("vimal");
list.add("ratan");
```

```
for(Object obj:list)
System.out.println(obj);

}}
```

@Deprecated

@Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

```
class A{
void m(){System.out.println("hello m");}

@Deprecated
void n(){System.out.println("hello n");}
}

class TestAnnotation3{
public static void main(String args[]){

A a=new A();
a.n();
}}
```

Example of custom annotation: creating, applying and accessing annotation

Built-in Annotations used in custom annotations in java

- @Target
- @Retention
- @Inherited
- @Documented

@Target

@Target tag is used to specify at which type, the annotation is used.

The java.lang.annotation.**ElementType** enum declares many constants to specify the type of element where annotation is to be applied such as TYPE, METHOD, FIELD etc. Let's see the constants of ElementType enum:

Element Types	Where the annotation can be applied
TYPE	class, interface or enumeration
FIELD	fields
METHOD	methods
CONSTRUCTOR	constructors
LOCAL_VARIABLE	local variables
ANNOTATION_TYPE	annotation type
PARAMETER	parameter

Example to specify annoation for a class

```
@Target(ElementType.TYPE)
@interface MyAnnotation{
int value1();
String value2();
}
```

Example to specify annotation for a class, methods or fields

```
@Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
@interface MyAnnotation{
int value1();
String value2();
}
```

@Retention

@Retention annotation is used to specify to what level annotation will be available.

RetentionPolicy	Availability
RetentionPolicy.SOURCE	refers to the source code, discarded during compilation. It will not be available in the compiled class.
RetentionPolicy.CLASS	refers to the .class file, available to java compiler but not to JVM . It is included in the class file.
RetentionPolicy.RUNTIME	refers to the runtime, available to java compiler and JVM .

Example to specify the RetentionPolicy

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@interface MyAnnotation{
int value1();
String value2();
}
```

```
//Creating annotation
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface MyAnnotation{
int value();
}

//Applying annotation
class Hello{
@MyAnnotation(value=10)
public void sayHello(){System.out.println("hello annotation");}
}

//Accessing annotation
class TestCustomAnnotation1{
public static void main(String args[])throws Exception{

Hello h=new Hello();
Method m=h.getClass().getMethod("sayHello");

MyAnnotation manno=m.getAnnotation(MyAnnotation.class);
System.out.println("value is: "+manno.value());
}}

//Output: value is 10
```

Servlets I Servlet Tutorial

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming

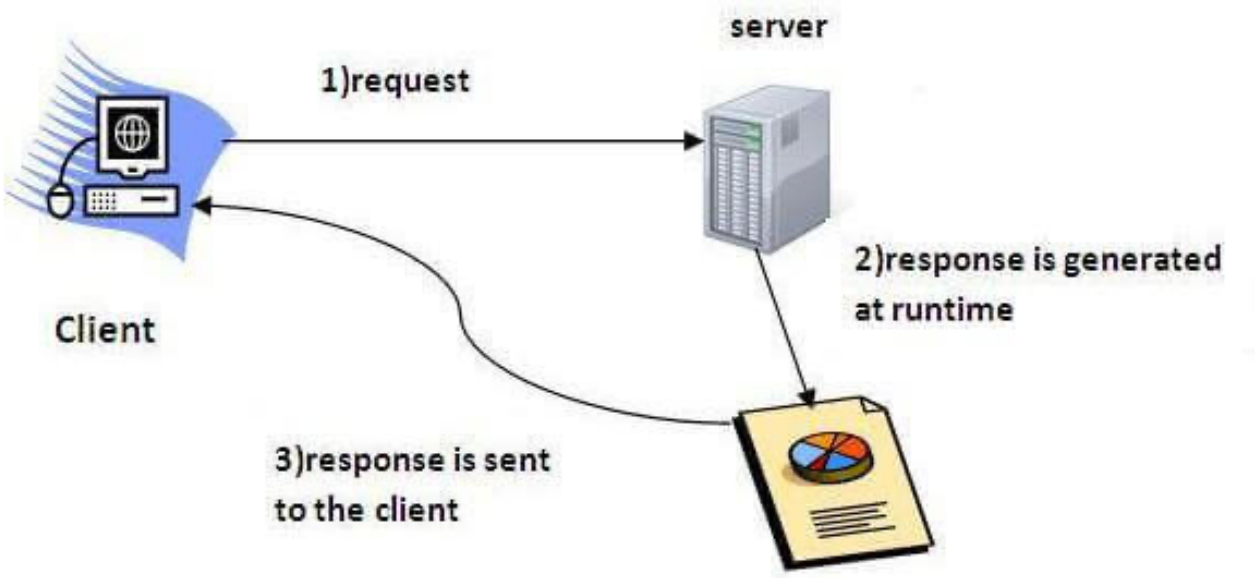
language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

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

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

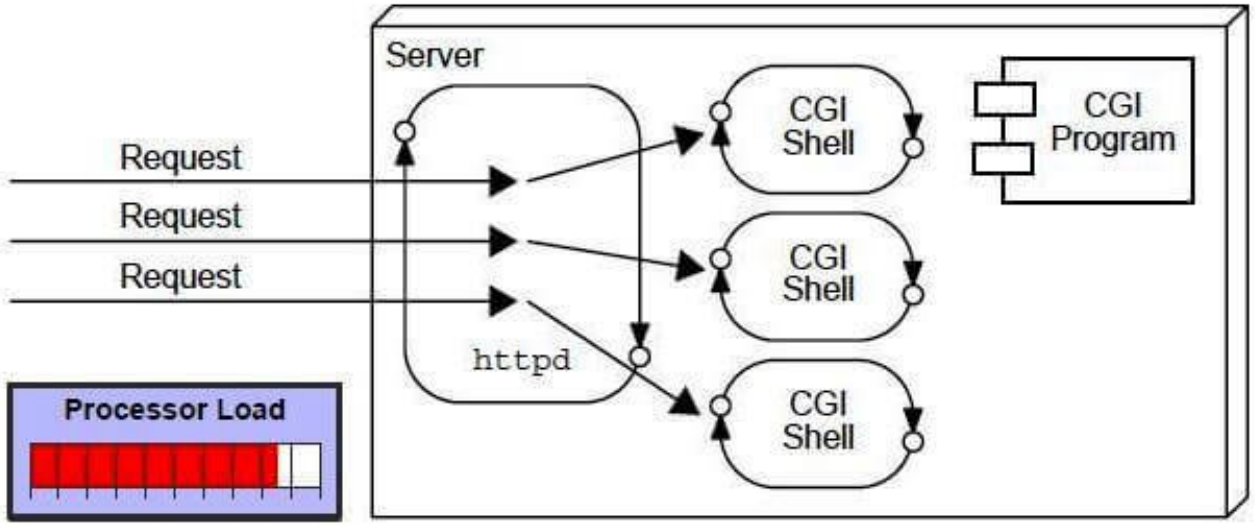


What is a web application?

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

CGI (Common Gateway Interface)

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

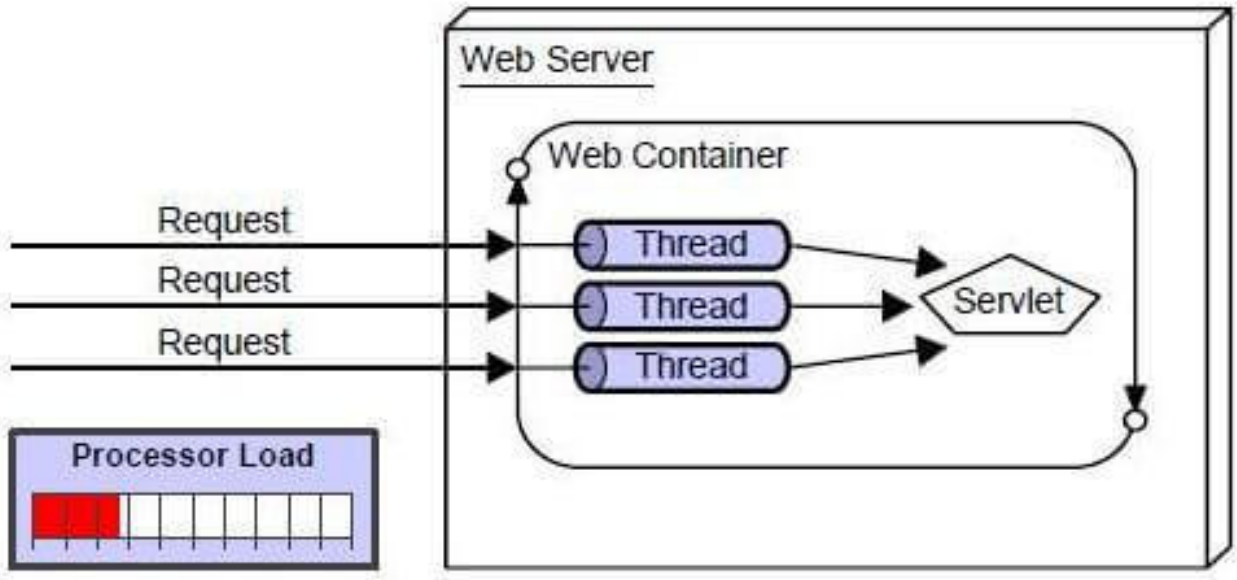


Disadvantages of CGI

There are many problems in CGI technology:

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

Advantages of Servlet



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

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

ATM program Java

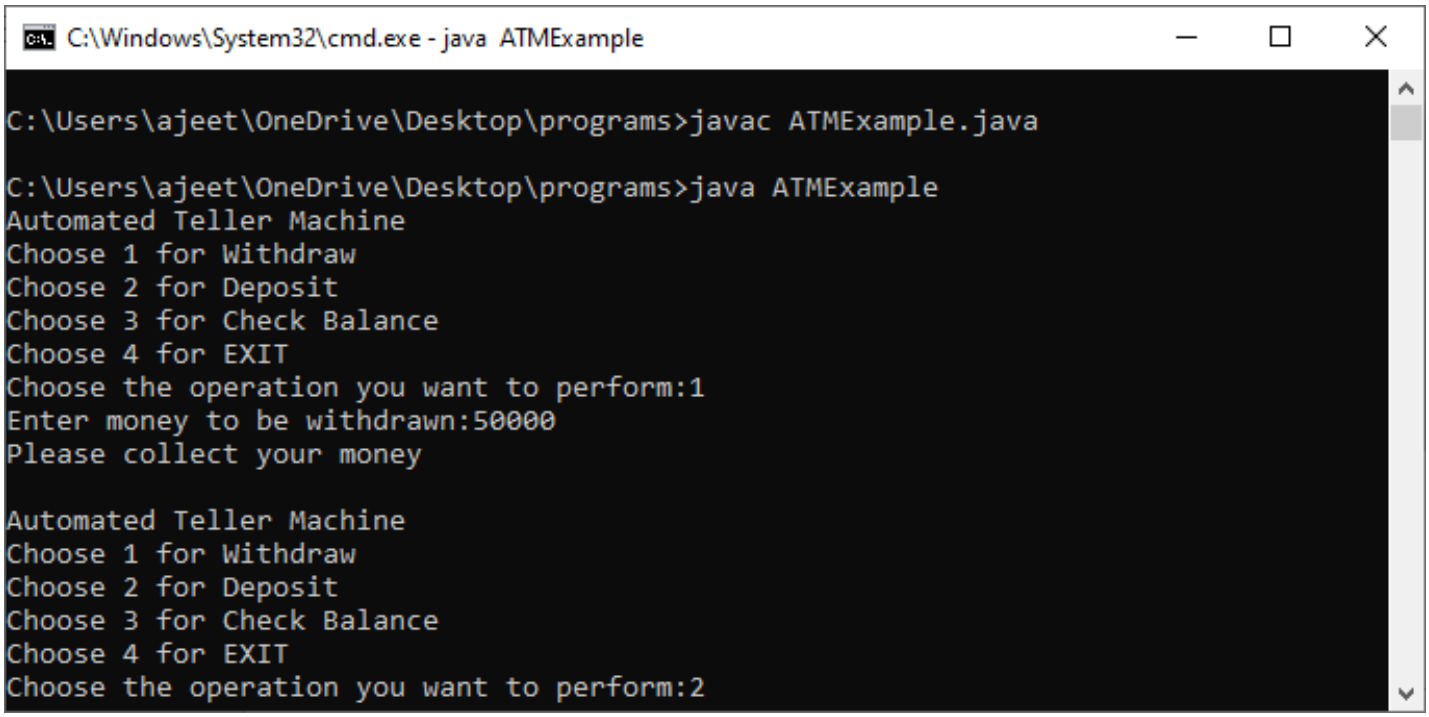
In Java, we can create an ATM program for representing ATM transection. In the ATM program, the user has to select an option from the options displayed on the screen. The options are related

to withdraw the money, deposit the money, check the balance, and exit.

To withdraw the money, we simply get the withdrawal amount from the user and remove that amount from the total balance and print the successful message.

To deposit the money, we simply get the deposit amount from the user, add it to the total balance and print the successful message.

We use the exit(0) method to exit from the current Transaction mode and return the user to the home page or initial screen.



Servlet API

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

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

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

Interfaces in javax.servlet package

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

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

Classes in javax.servlet package

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

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

Interfaces in javax.servlet.http package

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

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

Classes in javax.servlet.http package

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

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

Servlet interface provides common behaviorto 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 clas

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides

the implementation of all the methods of these interfaces except the service method.

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

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

Methods of GenericServlet class

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

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

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

Servlet Example by inheriting the GenericServlet class

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class First extends GenericServlet{
5. public void service(ServletRequest req,ServletResponse res)
6. throws IOException,ServletException{
7.
8. res.setContentType("text/html");
9.
10. PrintWriter out=res.getWriter();
11. out.print("<html><body>");
12. out.print("<b>hello generic servlet</b>");
13. out.print("</body></html>");
14.
15. }
16. }
```

HttpServlet class

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

Methods of HttpServlet class

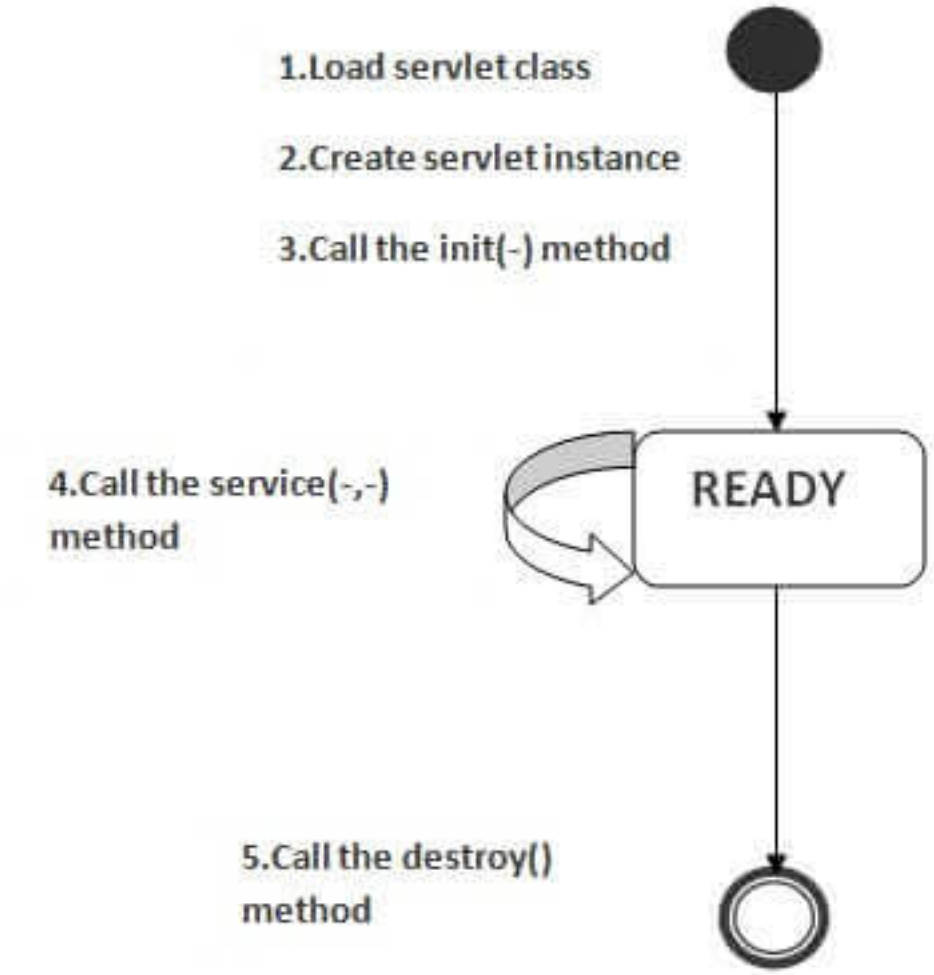
There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle)

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

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



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

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 method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
1. 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:

```
1. public void service(ServletRequest request, ServletResponse
   response)
2.   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:

```
1. public void destroy()
```

Steps to create a servlet example

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

The servlet example can be created by three ways:

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

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

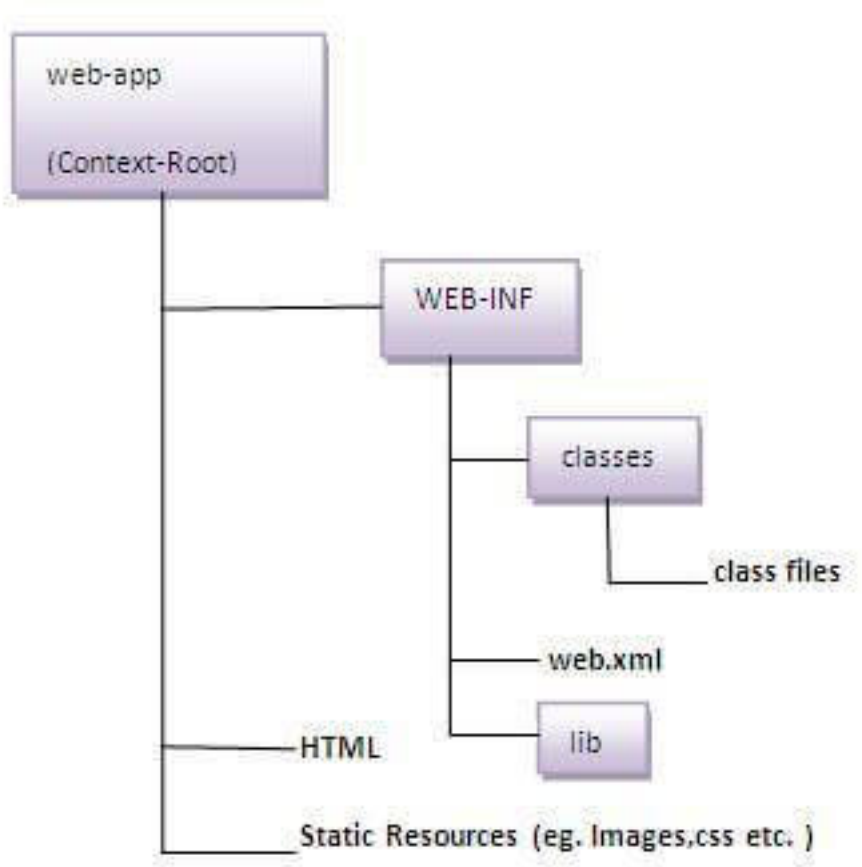
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

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

1)Create a directory structures

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

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



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

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

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

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

DemoServlet.java

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

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

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

3)Compile the servlet

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

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

Two ways to load the jar file

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

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

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

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

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

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

web.xml file

```
<web-app>

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

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

</web-app>
```

Description of the elements of web.xml file

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

<web-app> represents the whole application.
<servlet> is sub element of <web-app> and represents the servlet.
<servlet-name> is sub element of <servlet> represents the name of the servlet.
<servlet-class> is sub element of <servlet> represents the class of the servlet.
<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.
<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5)Start the Server and deploy the project

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

One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

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

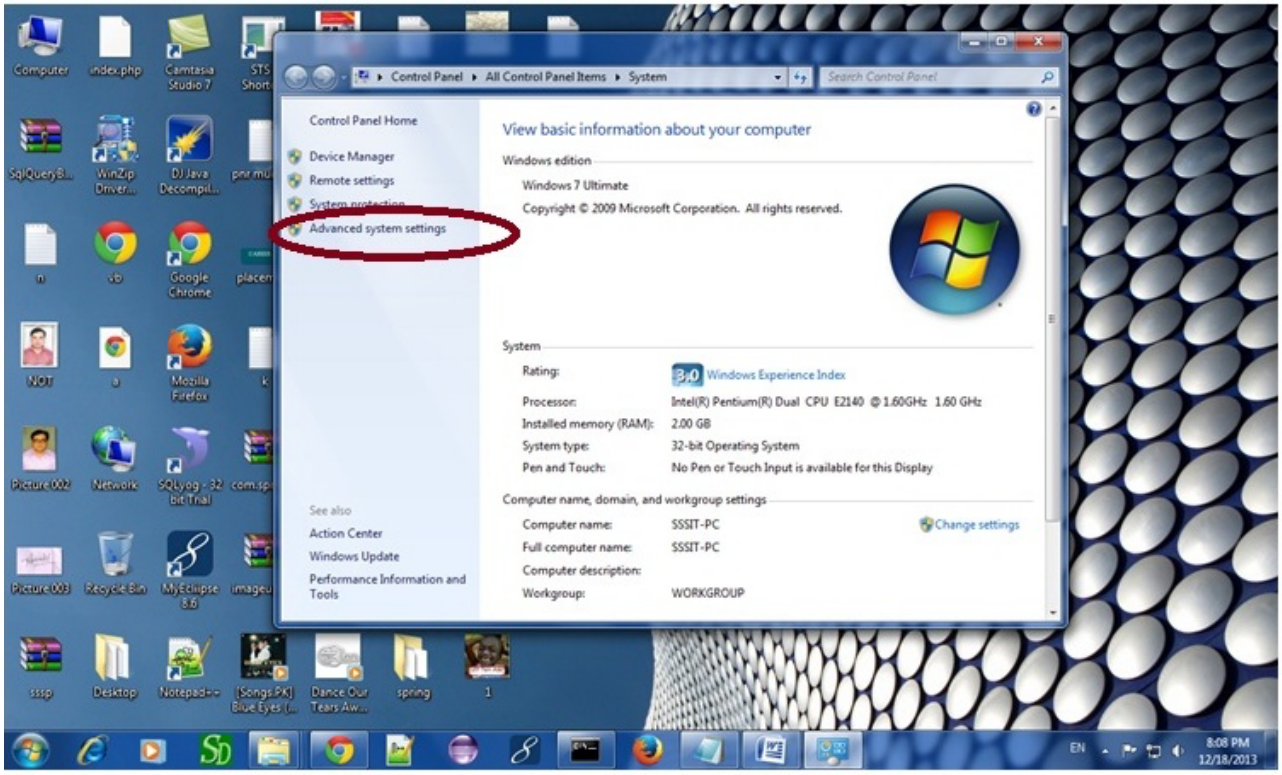
1) How to set JAVA_HOME in environment variable?

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

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

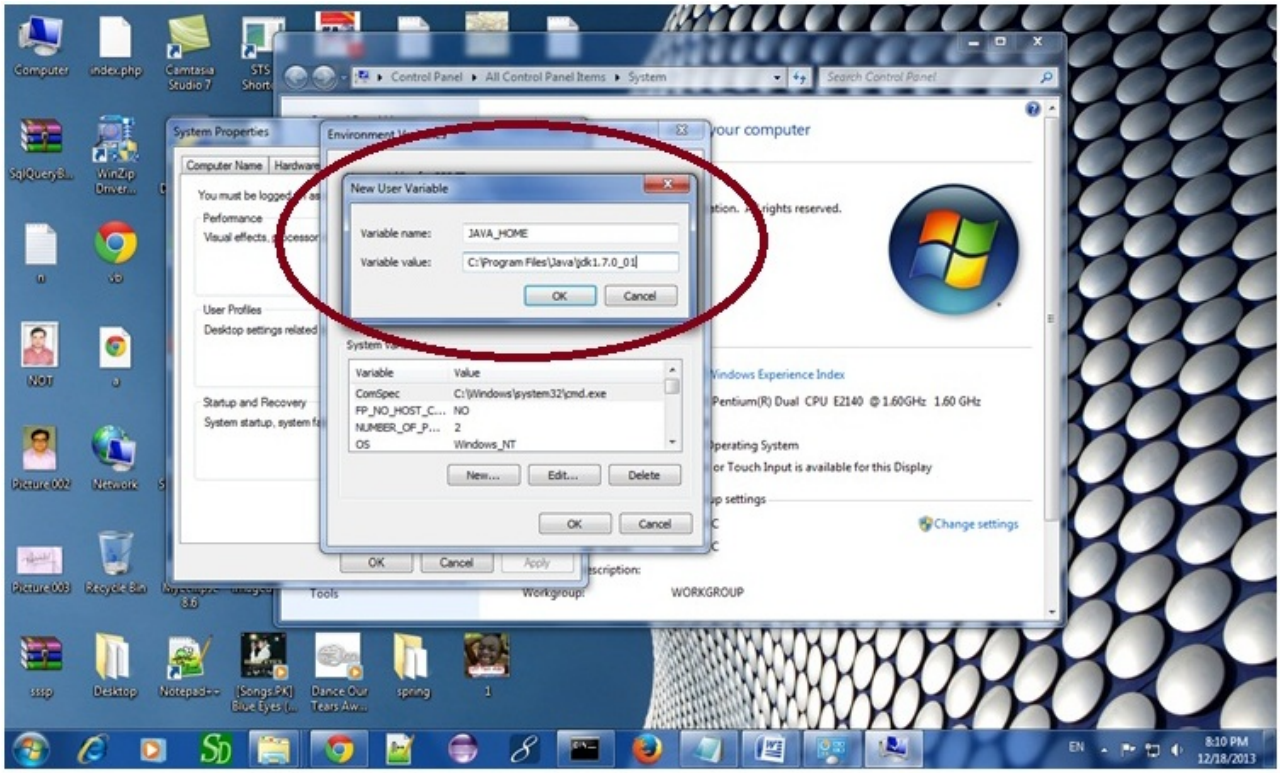
Go to My Computer properties:

Click on advanced system settings tab then environment variables:



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

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

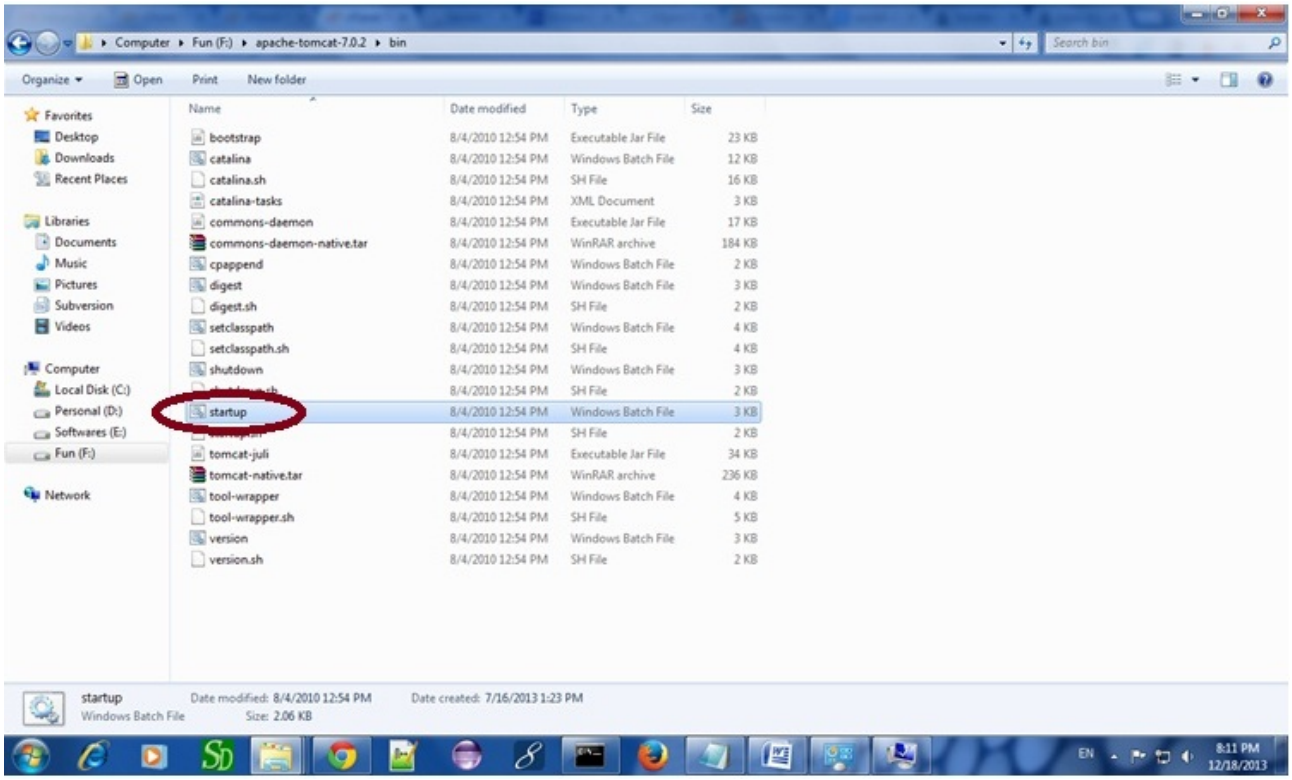


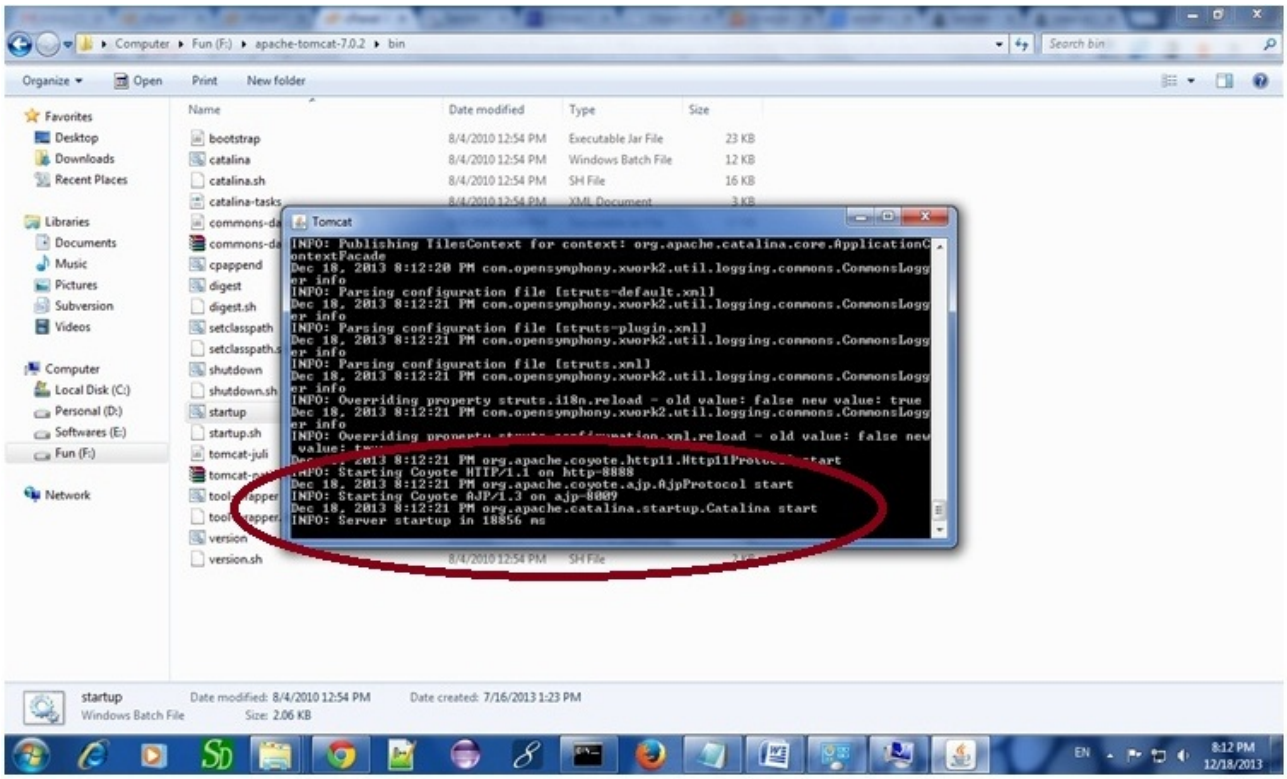
There must not be semicolon (;) at the end of the path.

After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.
Note: There are two types of tomcat available:

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

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





Now server is started successfully.

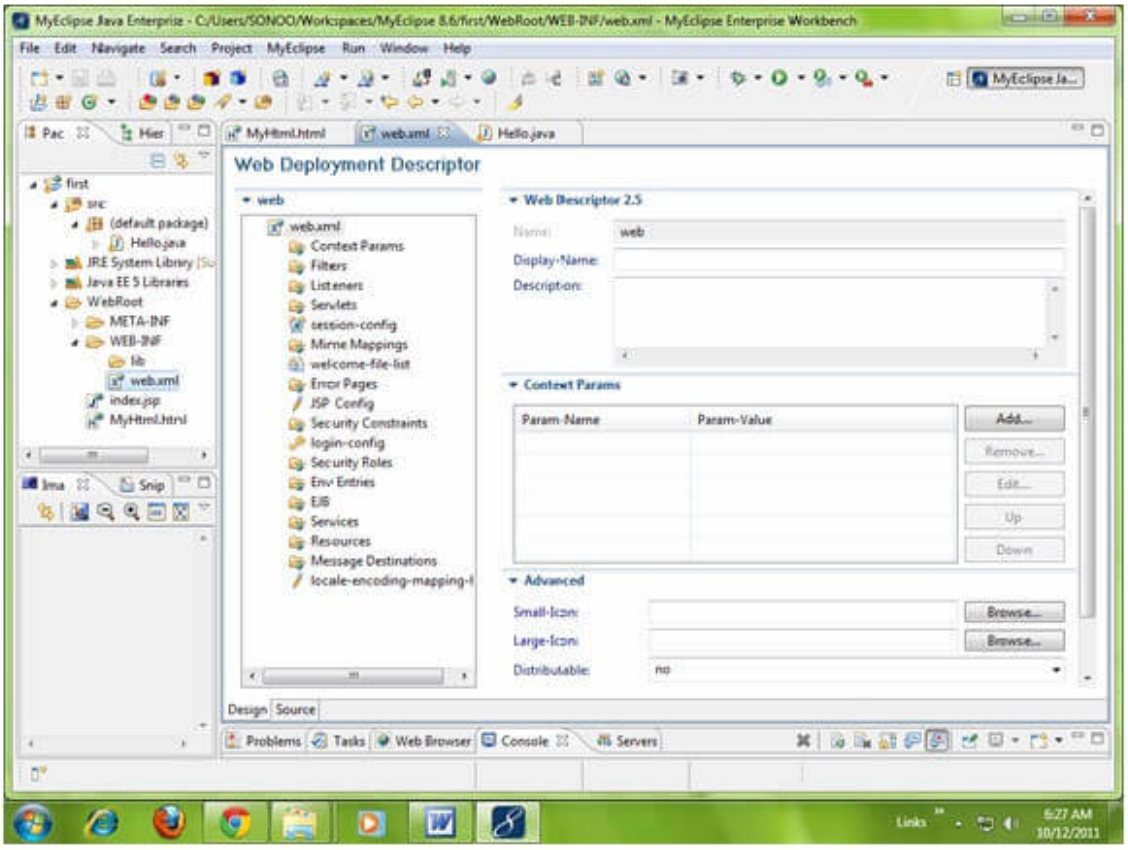
2) How to change port number of apache tomcat

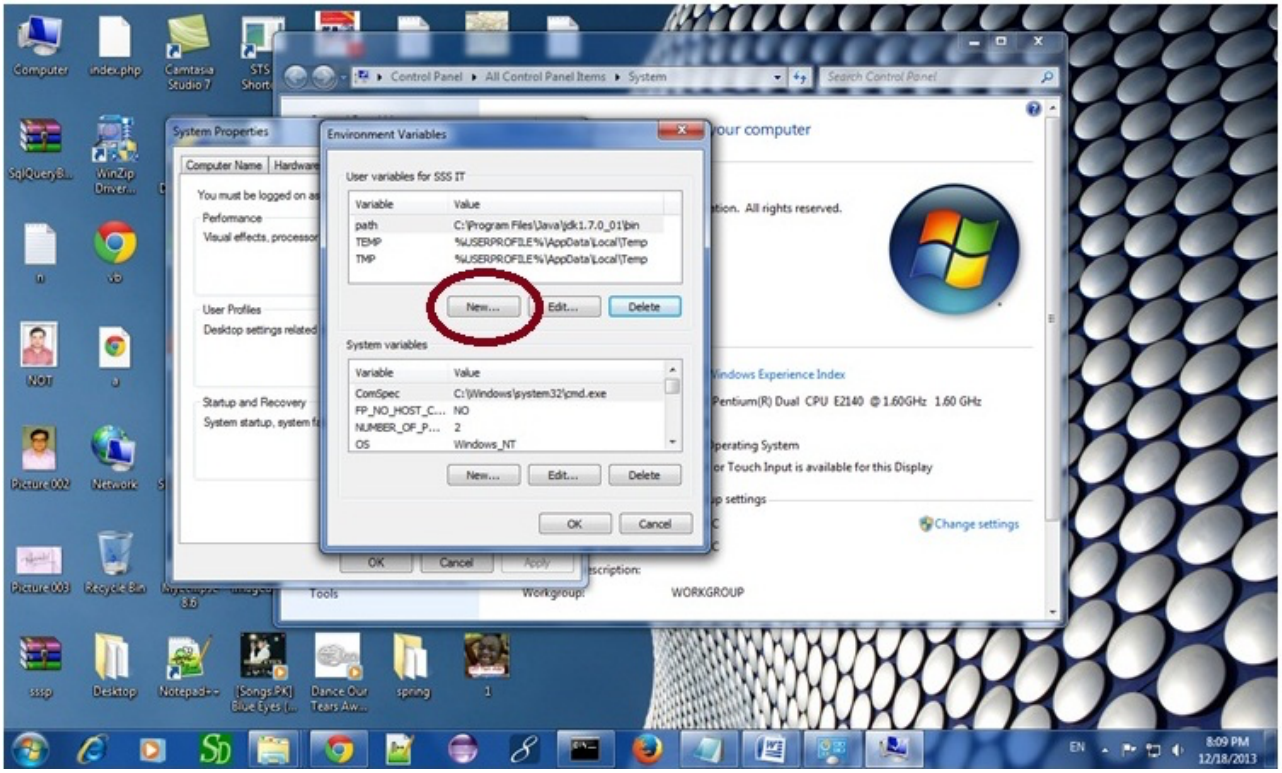
Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

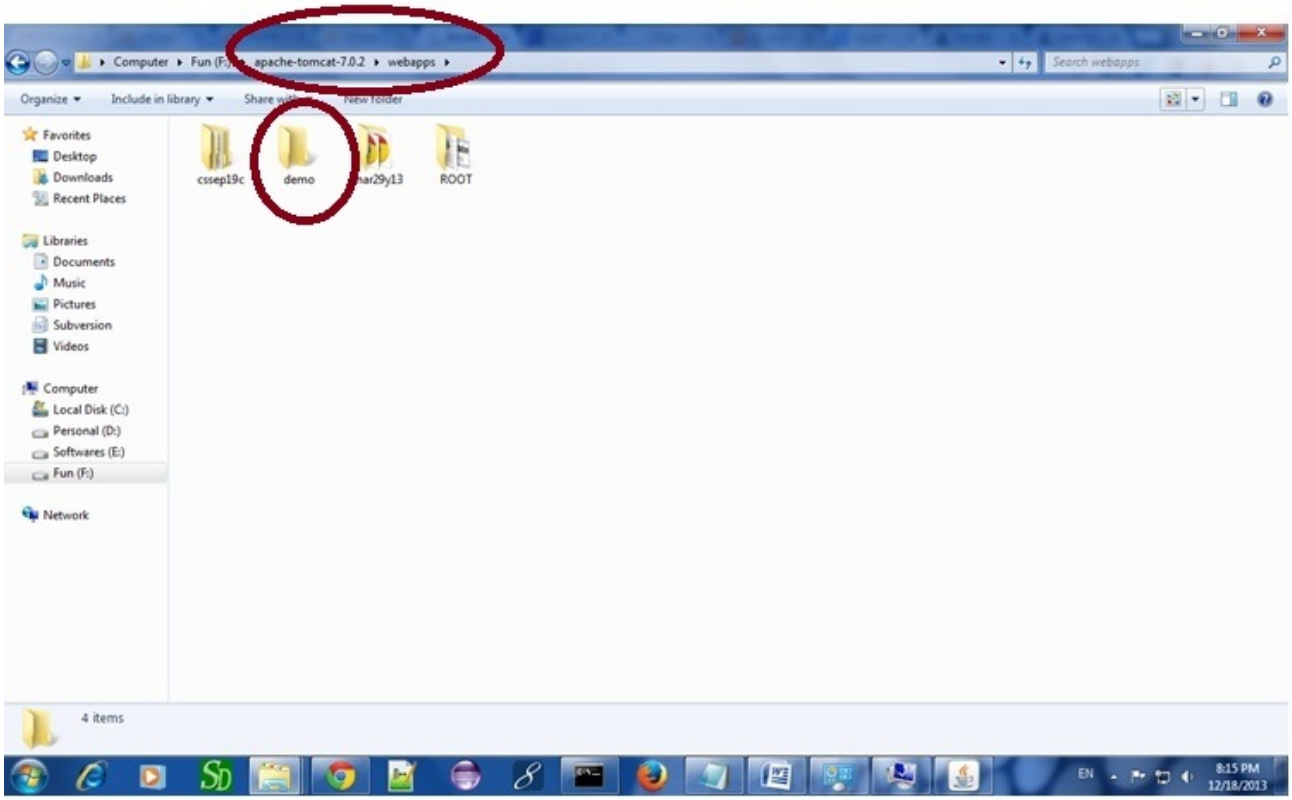
Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

5) How to deploy the servlet project

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

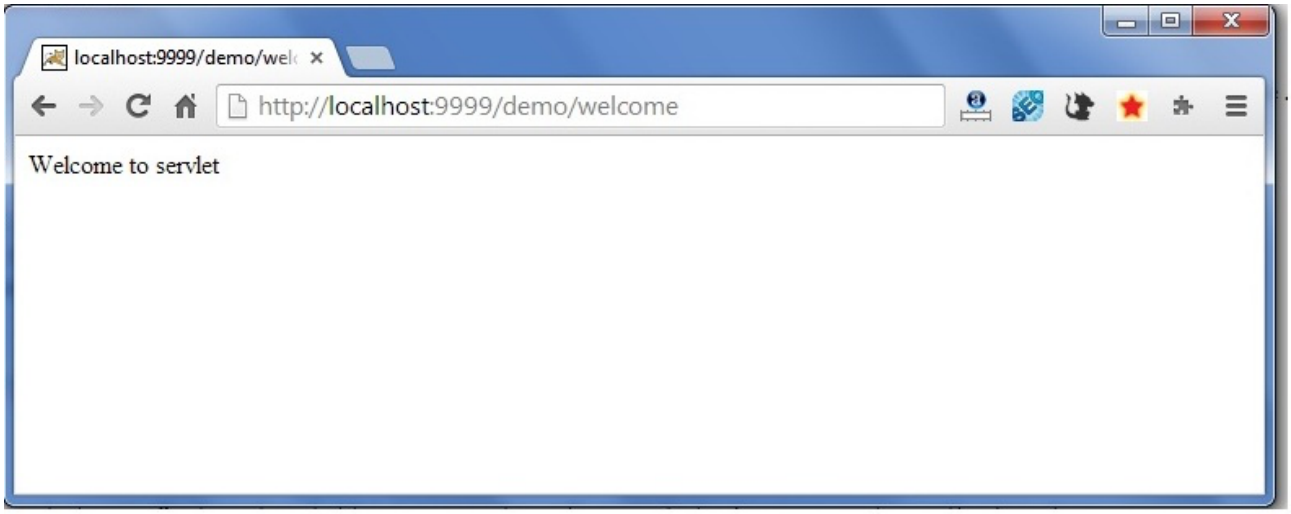






But there are several ways to deploy the project. They are as follows:

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory



- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

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

```
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
```

How Servlet works?

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

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

If yes, web container does the following tasks:

- loads the servlet class.
- instantiates the servlet class.
- calls the init method passing the ServletConfig object

else

- calls the service method passing request and response objects

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

How web container handles the servlet request?

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

- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

What is written inside the public service method?

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

```
1.  public void service(ServletRequest req, ServletResponse res)
2.          throws ServletException, IOException
3.     {
4.          HttpServletRequest request;
5.          HttpServletResponse response;
6.          try
7.          {
8.              request = (HttpServletRequest)req;
9.              response = (HttpServletResponse)res;
10.         }
11.         catch(ClassCastException e)
12.         {
13.                                 throw new ServletException("non-
    HTTP request or response");
14.         }
15.         service(request, response);
16.    }
```

What is written inside the protected service method?

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

```
1.  protected void service(HttpServletRequest req, HttpServletR
    esponse resp)
2.          throws ServletException, IOException
3.     {
4.          String method = req.getMethod();
5.          if(method.equals("GET"))
6.          {
7.              long lastModified = getLastModified(req);
8.              if(lastModified == -1L)
9.              {
10.                 doGet(req, resp);
```

```
11.         }
12.     ....
13.     //rest of the code
14.         }
15.     }
```

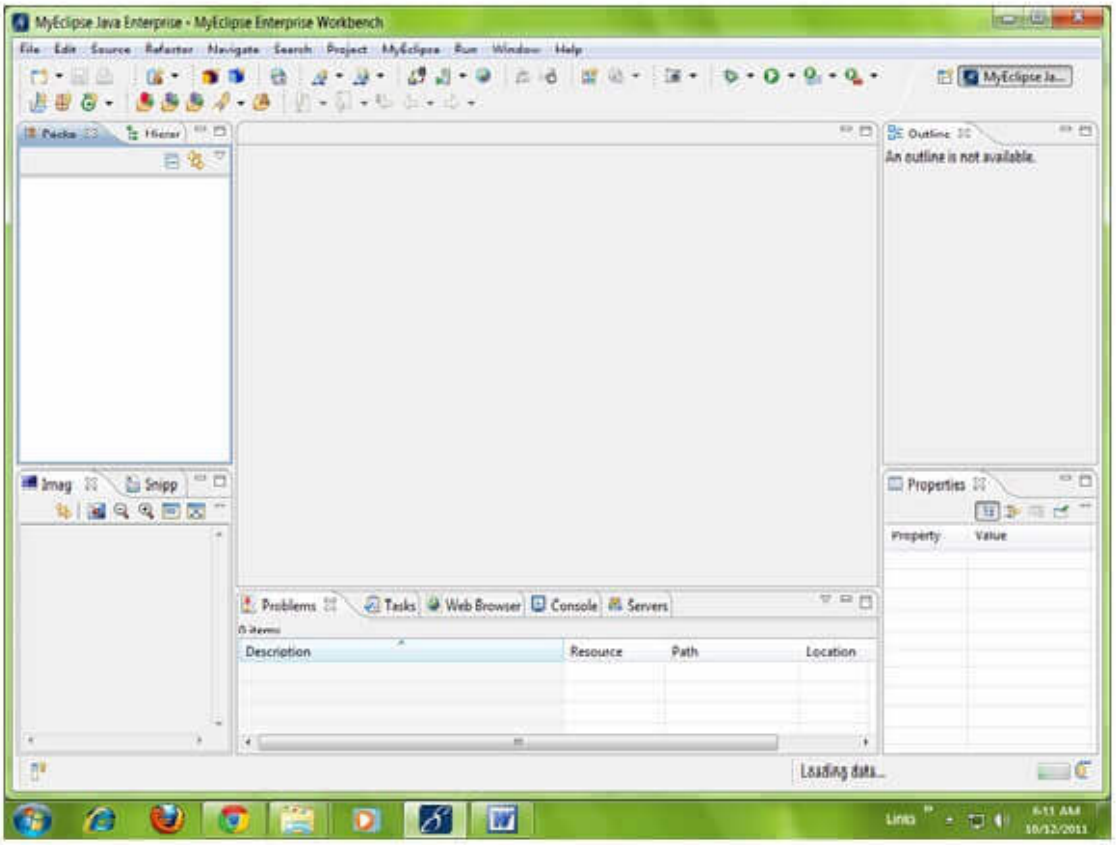
Creating Servlet in myeclipse IDE

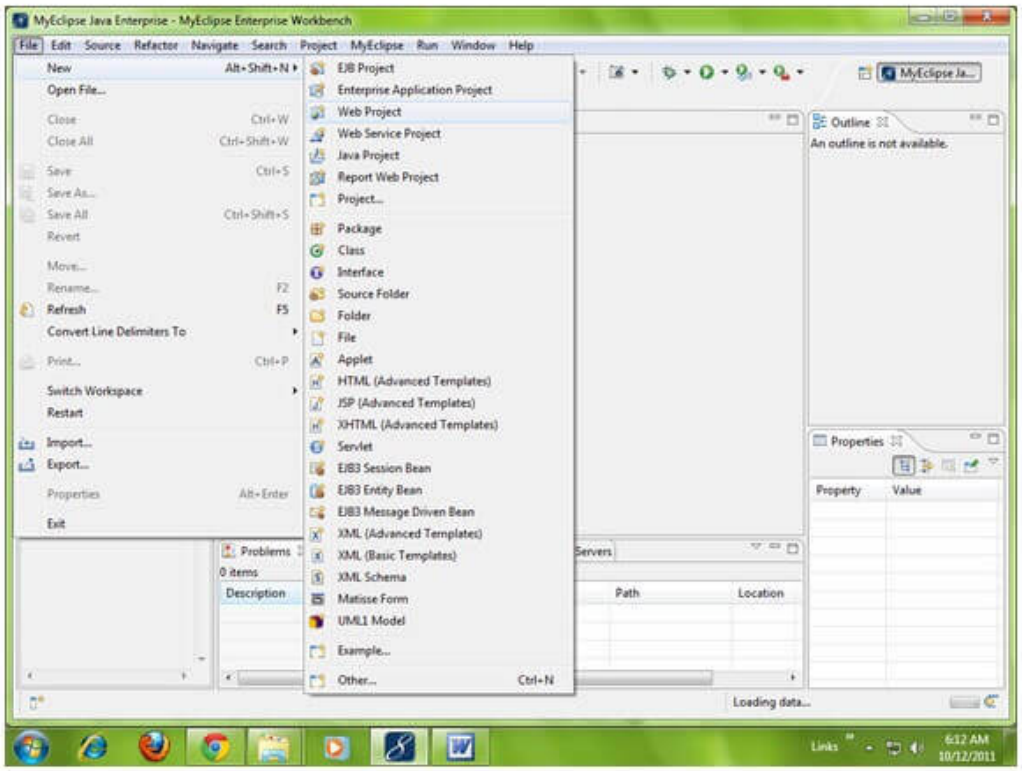
You need to follow the following steps to create the servlet in the myeclipse IDE. The steps are as follows:

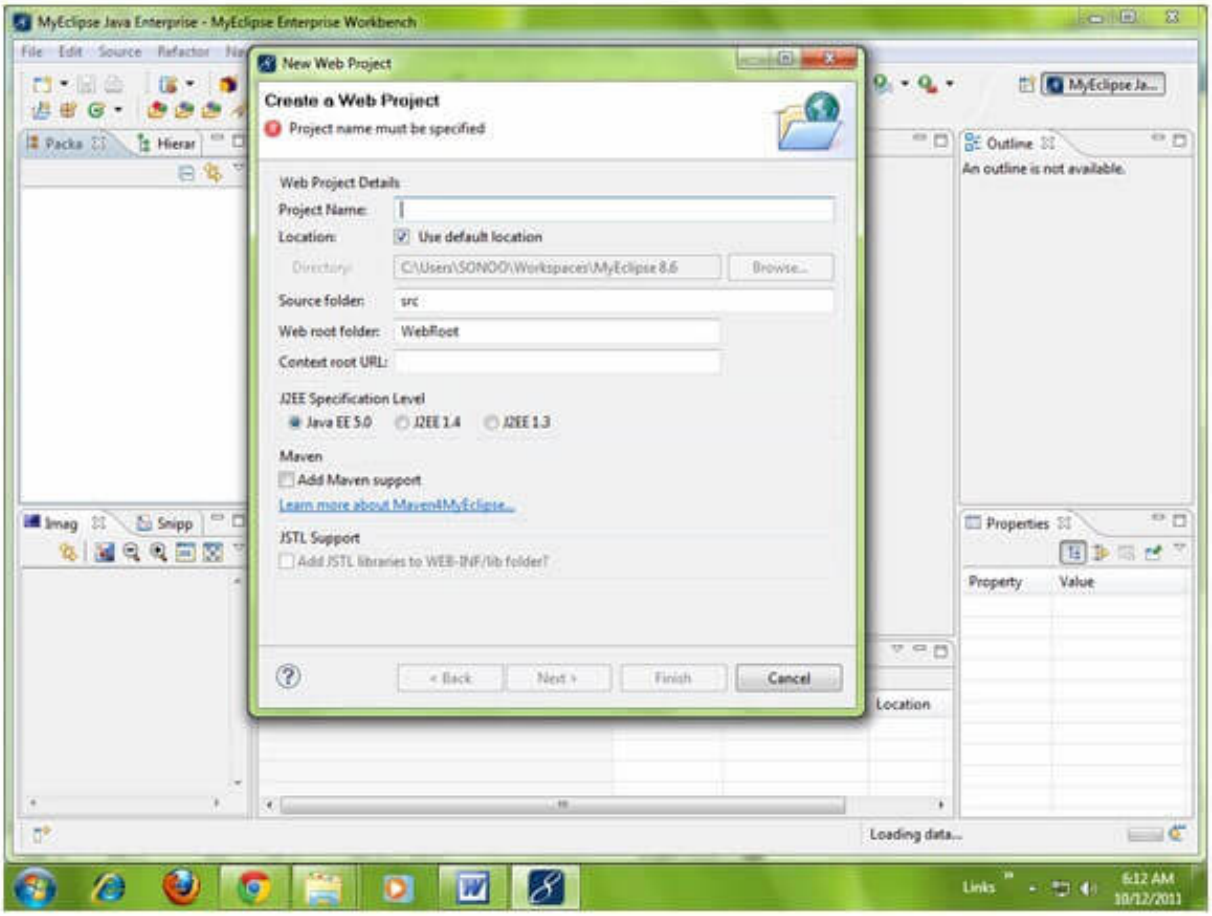
- Create a web project
- create a html file
- create a servlet
- start myeclipse tomcat server and deploy project

1) Create the web project:

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

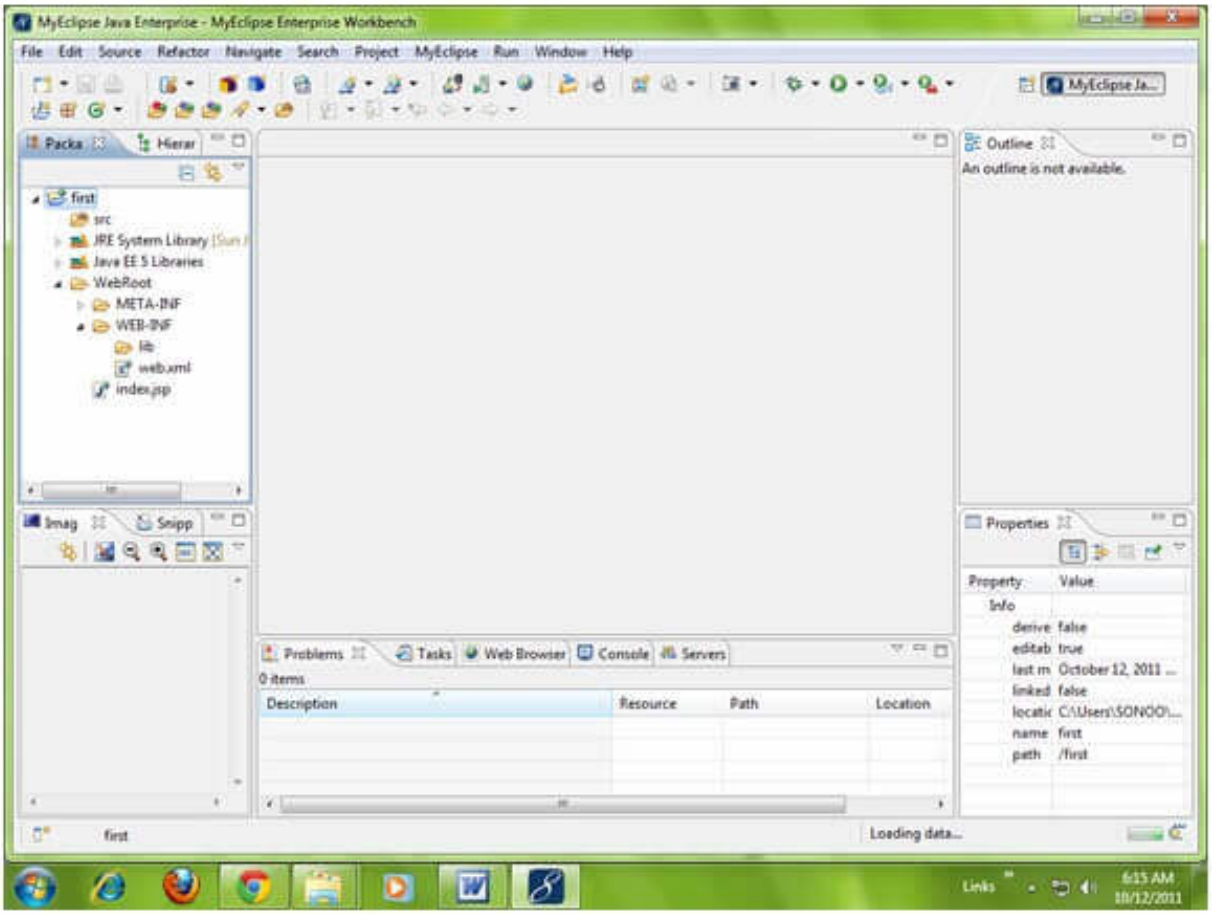



MyEclipse Java Enterprise - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
New Alt+Shift+N
Open File...
Close Ctrl+W
Close All Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters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

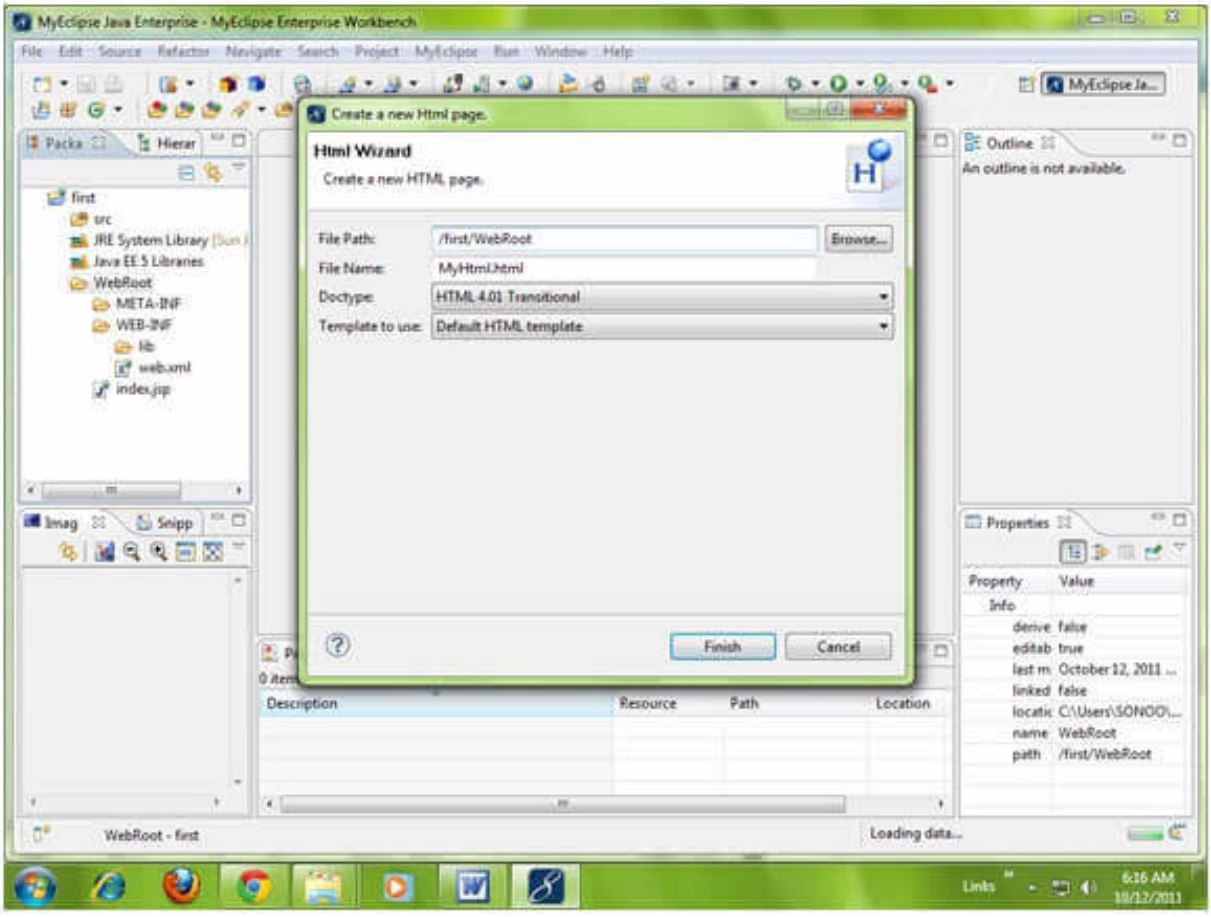

MyEclipse Java Enterprise - MyEclipse Enterprise Workbench
New Web Project
Create a Web Project
Project name must be specified
Web Project Details
Project Name:
Location:
Use default location
Directory:
C:\Users\SONOO\Workspaces\MyEclipse 8.6
Browse...
Source folder:
src
Web root folder:
WebRoot
Context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

2) Create the html file:

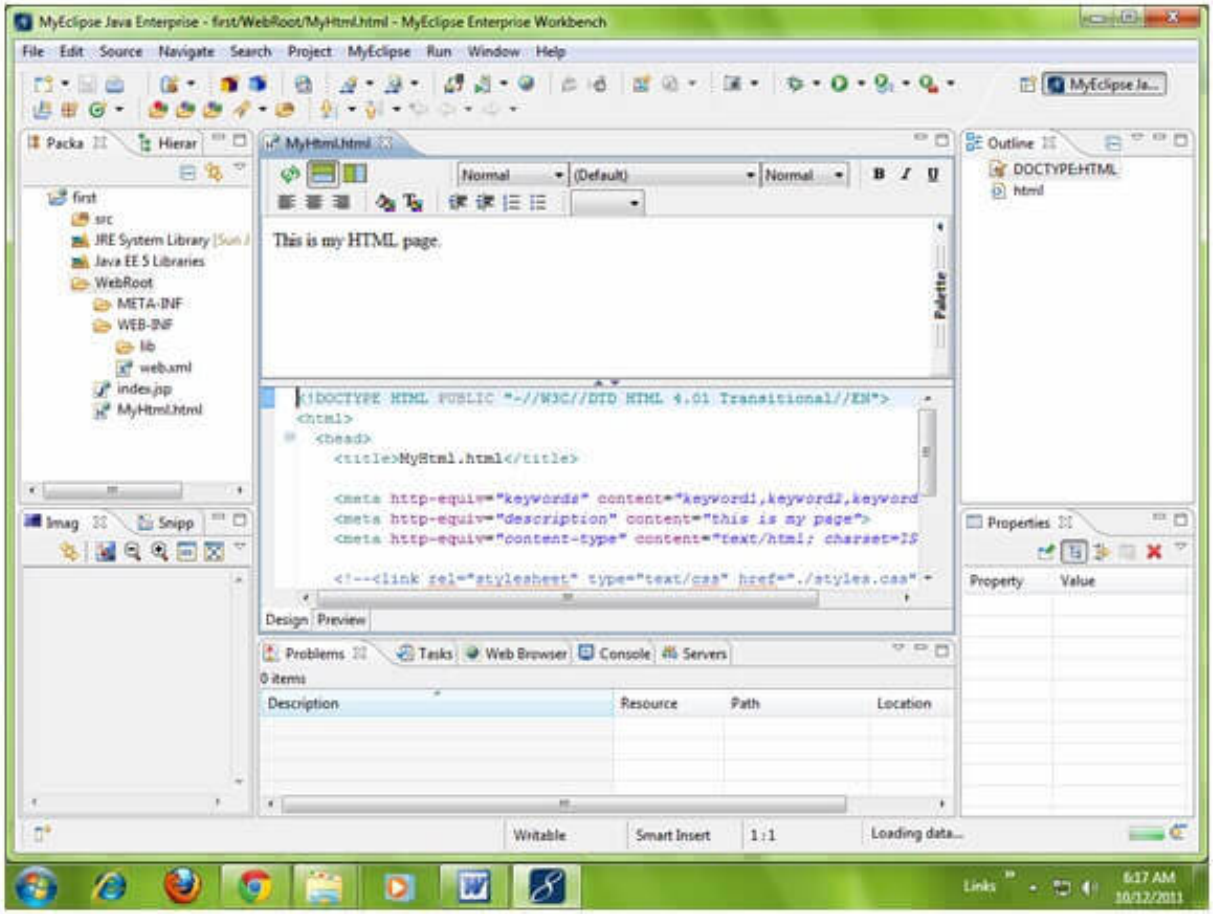
As you can see that a project is created named first. Now let's explore this project.

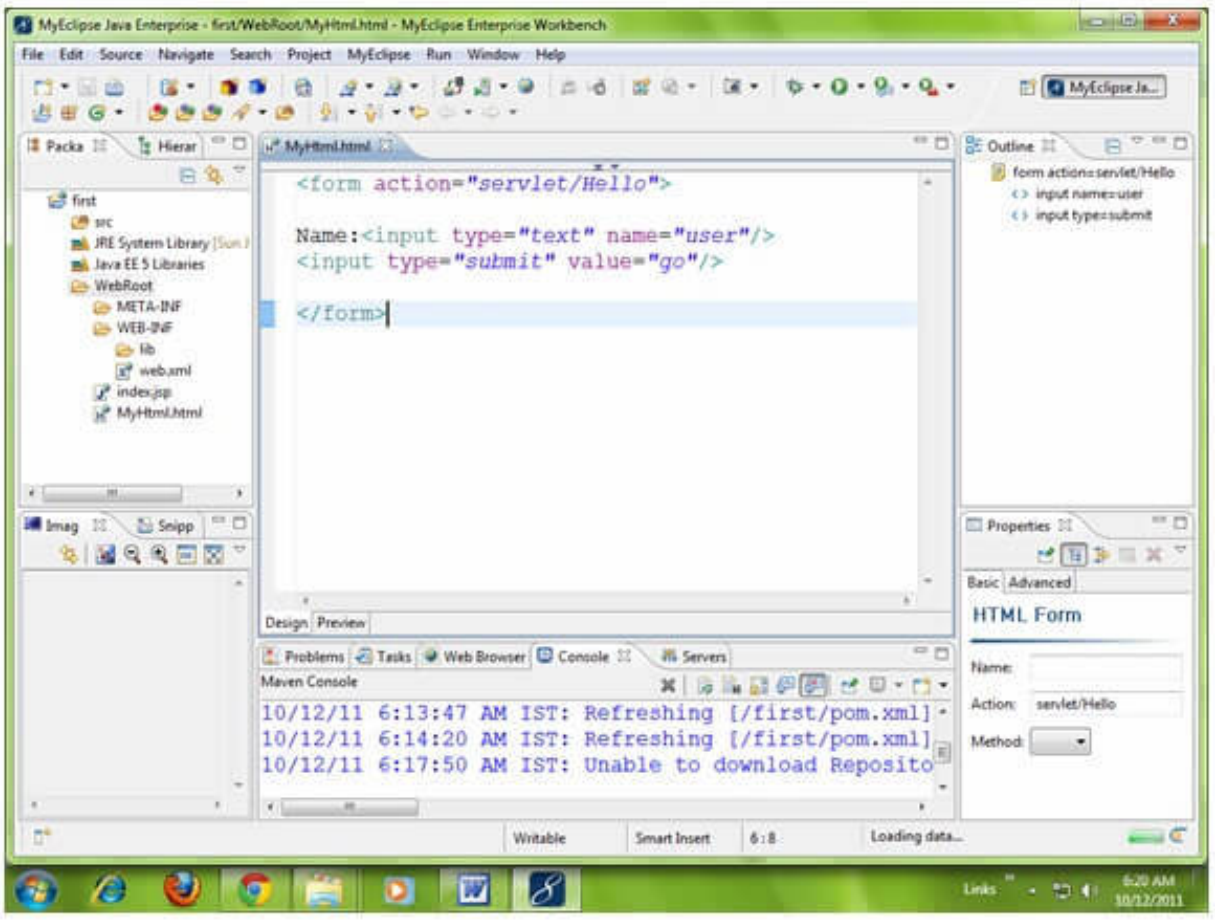


For creating a html file, right click on WebRoot -> New -> html -> write your html file name e.g. MyHtml.html -> Finish.



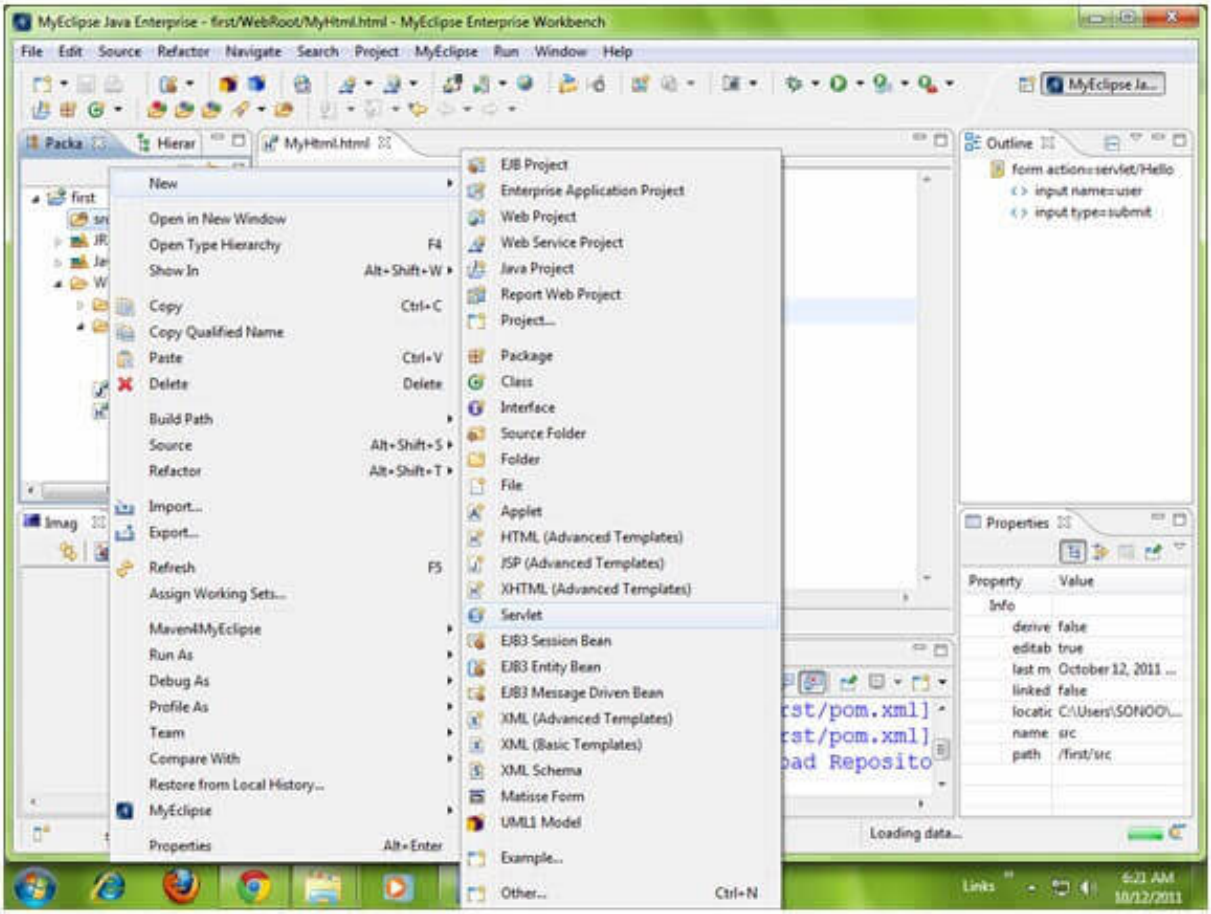
As you can see that a html file is created named MyHtml.html. Now let's write the html code here.

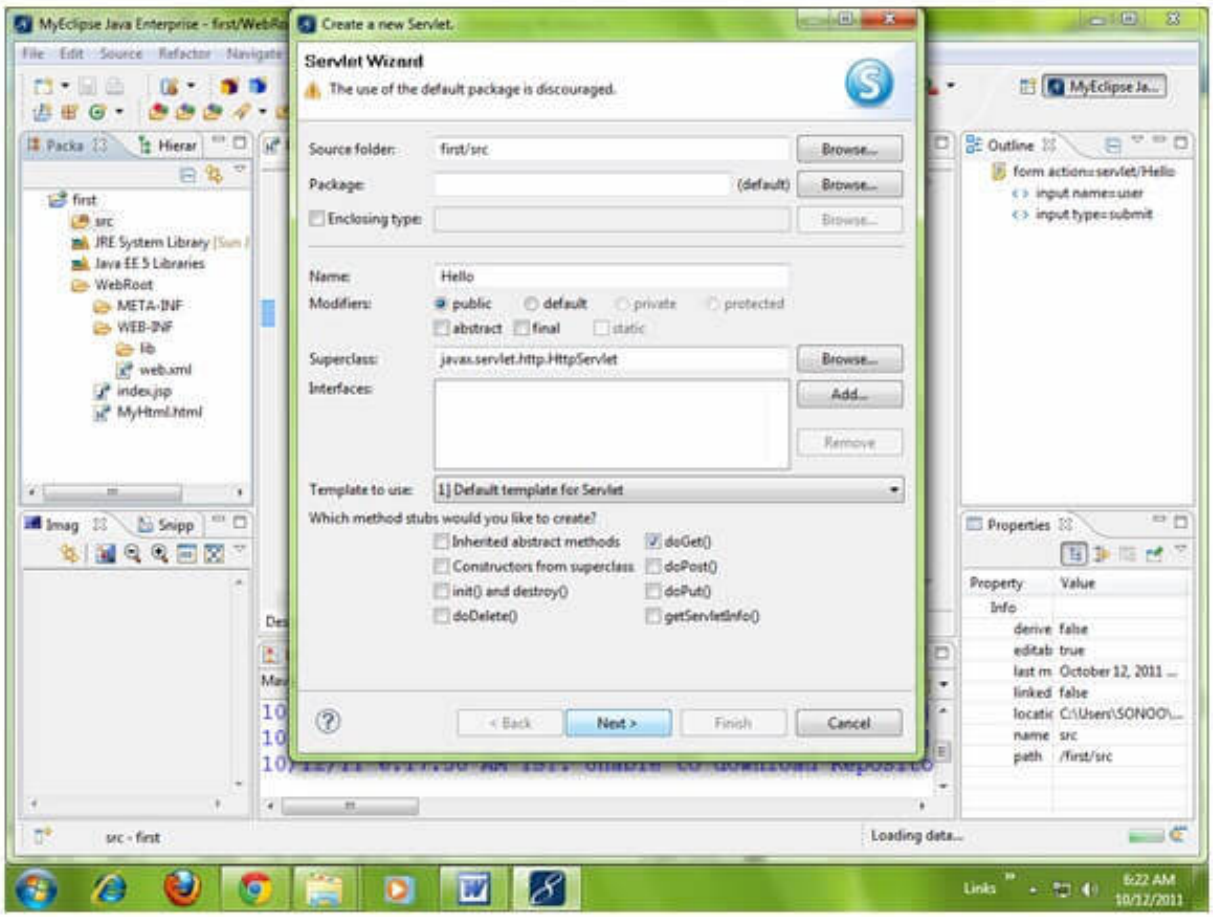

MyEclipse Java Enterprise - first/WebRoot/MyHtml.html - MyEclipse Enterprise Workbench
File Edit Source Navigate Search Project MyEclipse Run Window Help
Packa Hierar
first
src
JRE System Library
Java EE 5 Libraries
WebRoot
META-INF
WEB-INF
lib
web.xml
index.jsp
MyHtml.html
MyHtml.html
Normal
(Default)
Normal
This is my HTML page.
Palette
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>MyHtml.html</title>
<meta http-equiv="keywords" content="keyword1,keyword2,keyword
<meta http-equiv="description" content="this is my page">
<meta http-equiv="content-type" content="text/html; charset=IS
<!--<link rel="stylesheet" type="text/css" href="./styles.css"
Design Preview
Outline
DOCTYPE:HTML
html
Properties
Property Value
Problems Tasks Web Browser Console Servers
0 items
Description Resource Path Location
Writable Smart Insert 1 : 1 Loading data...
Links
6:17 AM
10/12/2011

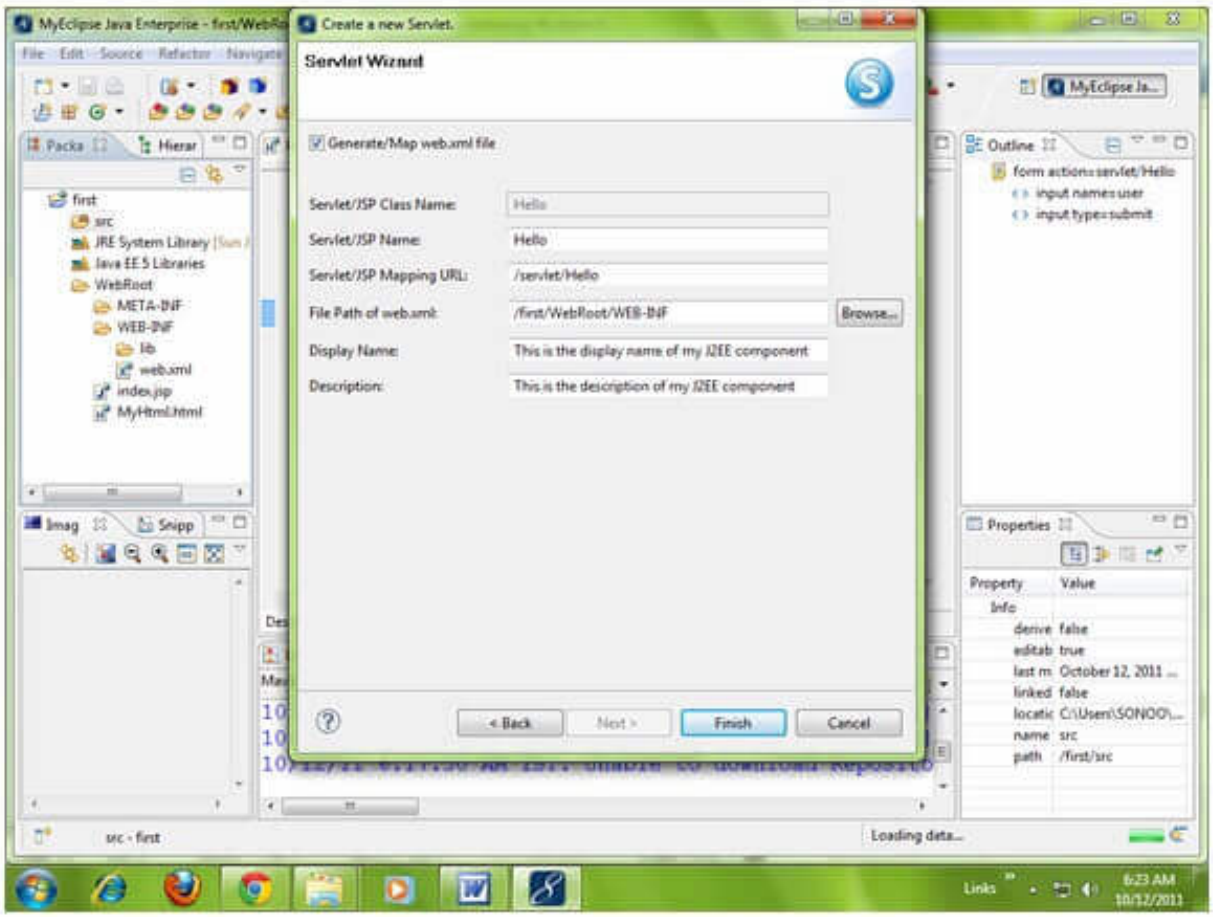


3) Create the servlet:

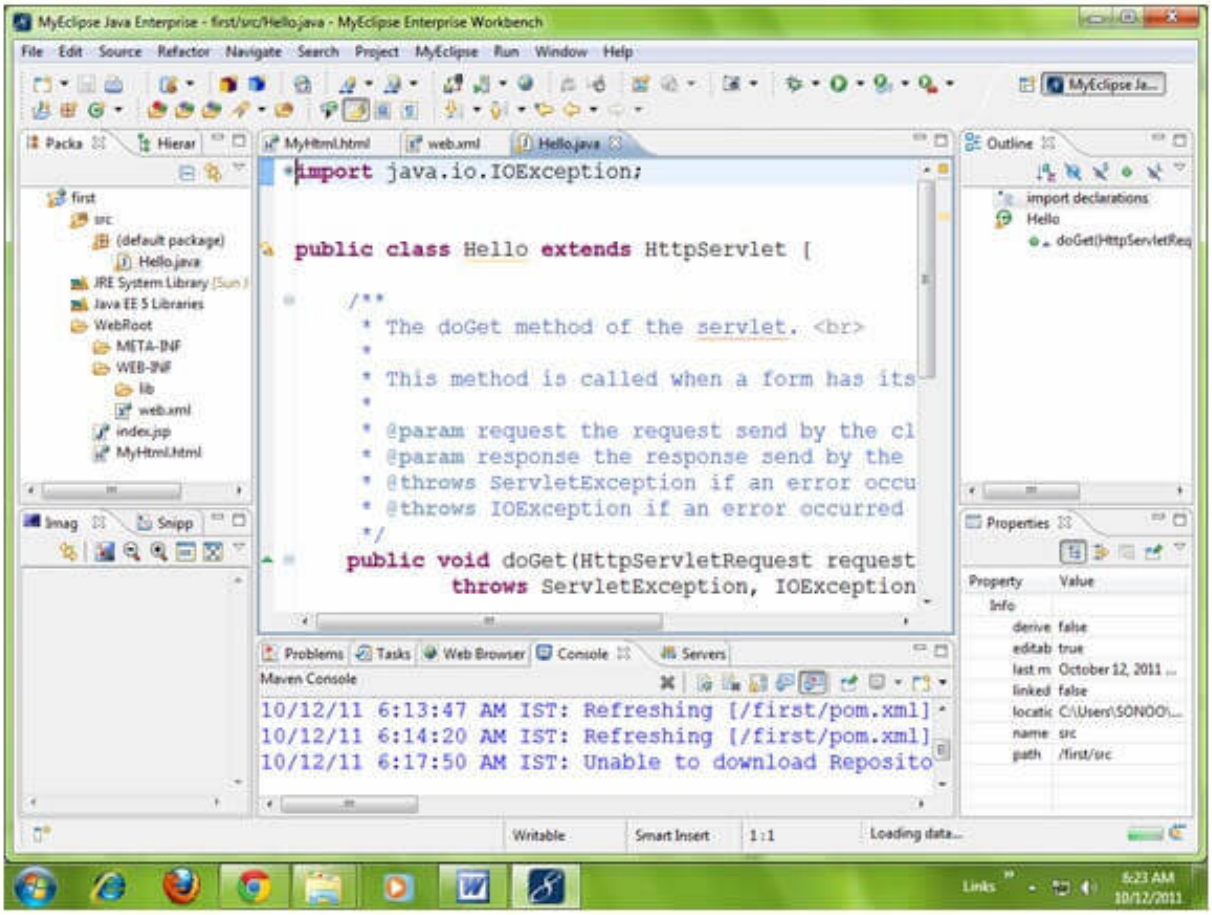
For creating a servlet click on File Menu -> New -> servlet -> write your servlet name e.g. Hello -> uncheck all the checkboxes except doGet() -> next -> Finish.

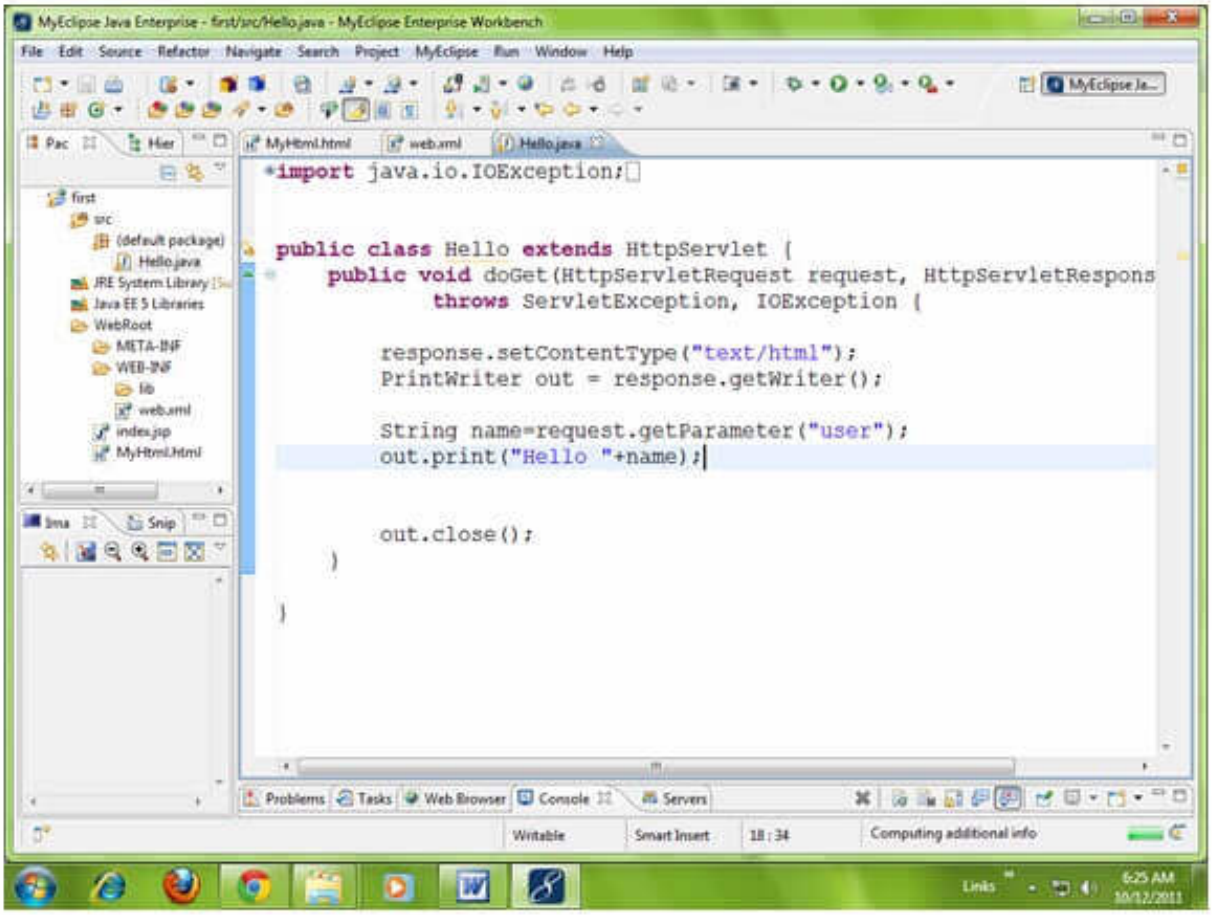

MyEclipse Java Enterprise - first/WebRoot/MyHtml.html - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
MyEclipse Ja...
Packa
Hierar
MyHtml.html
Outline
form action=servlet/Hello
input name=user
input type=submit
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Maven4MyEclipse
Run As
Debug As
Profile As
Team
Compare With
Restore from Local History...
MyEclipse
Properties Alt+Enter
EJB Project
Enterprise Application Project
Web Project
Web Service Project
Java Project
Report Web Project
Project...
Package
Class
Interface
Source Folder
Folder
File
Applet
HTML (Advanced Templates)
JSP (Advanced Templates)
XHTML (Advanced Templates)
Servlet
EJB3 Session Bean
EJB3 Entity Bean
EJB3 Message Driven Bean
XML (Advanced Templates)
XML (Basic Templates)
XML Schema
Matisse Form
UML1 Model
Example...
Other... Ctrl+N
Properties
Property Value
Info
derive false
editab true
last m October 12, 2011 ...
linked false
locatic C:\Users\SONOO\...
name src
path /first/src
Loading data...
Links
6:21 AM
10/12/2011


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

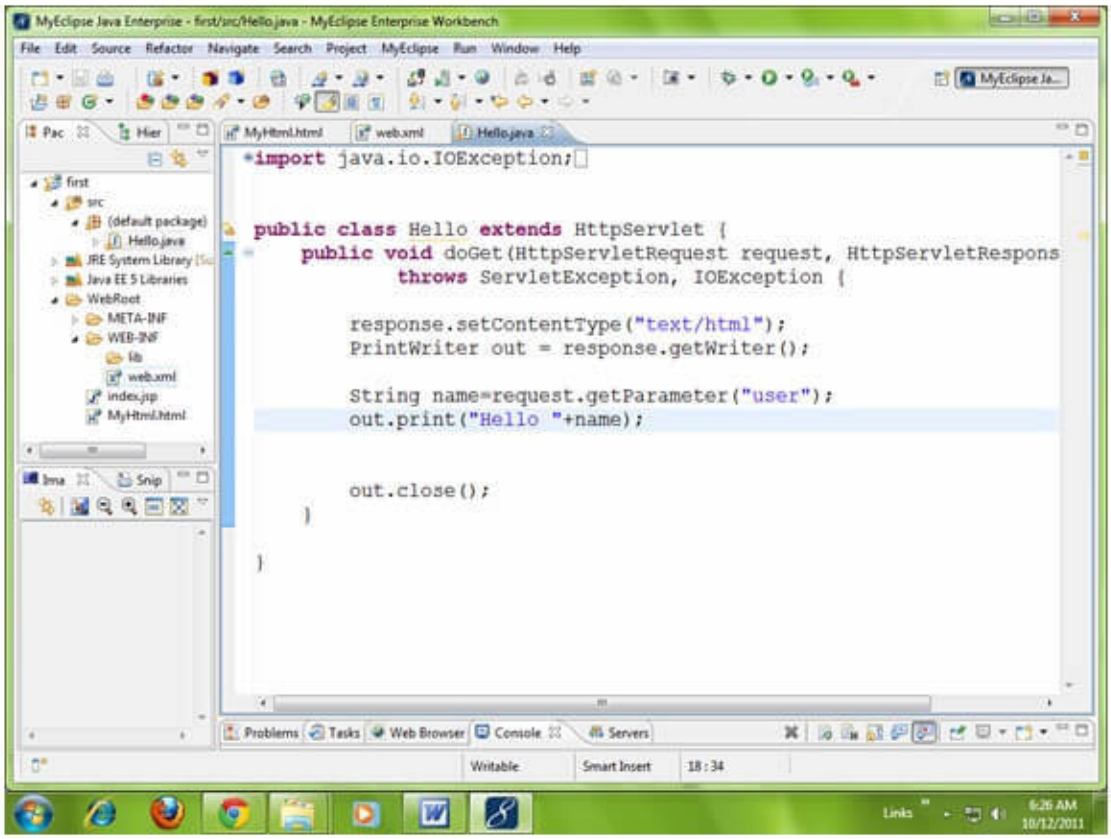


As you can see that a servlet file is created named Hello.java. Now let's write the servlet code here.


MyEclipse Java Enterprise - first/src/Hello.java - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
MyHtml.html
web.xml
Hello.java
*import java.io.IOException;
public class Hello extends HttpServlet {
/**
* The doGet method of the servlet.
*
* This method is called when a form has its
*
* @param request the request send by the cl
* @param response the response send by the
* @throws ServletException if an error occu
* @throws IOException if an error occurred
*/
public void doGet(HttpServletRequest request
throws ServletException, IOException
Outline
import declarations
Hello
doGet(HttpServletReq
Properties
Property Value
Info
derive false
editab true
last m October 12, 2011 ...
linked false
locatic C:\Users\SONOO\...
name src
path /first/src
Problems Tasks Web Browser Console Servers
Maven Console
10/12/11 6:13:47 AM IST: Refreshing [/first/pom.xml]
10/12/11 6:14:20 AM IST: Refreshing [/first/pom.xml]
10/12/11 6:17:50 AM IST: Unable to download Reposito
Writable
Smart Insert
1:1
Loading data...
Links
6:23 AM
10/12/2011

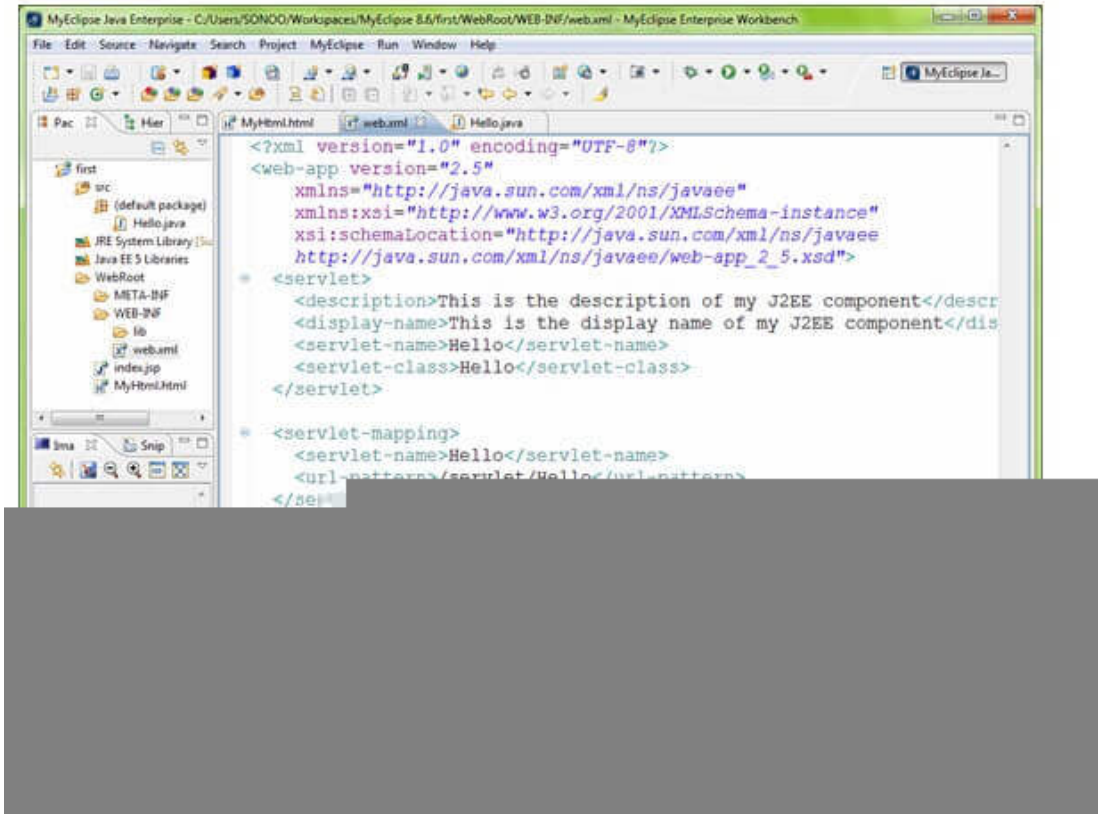


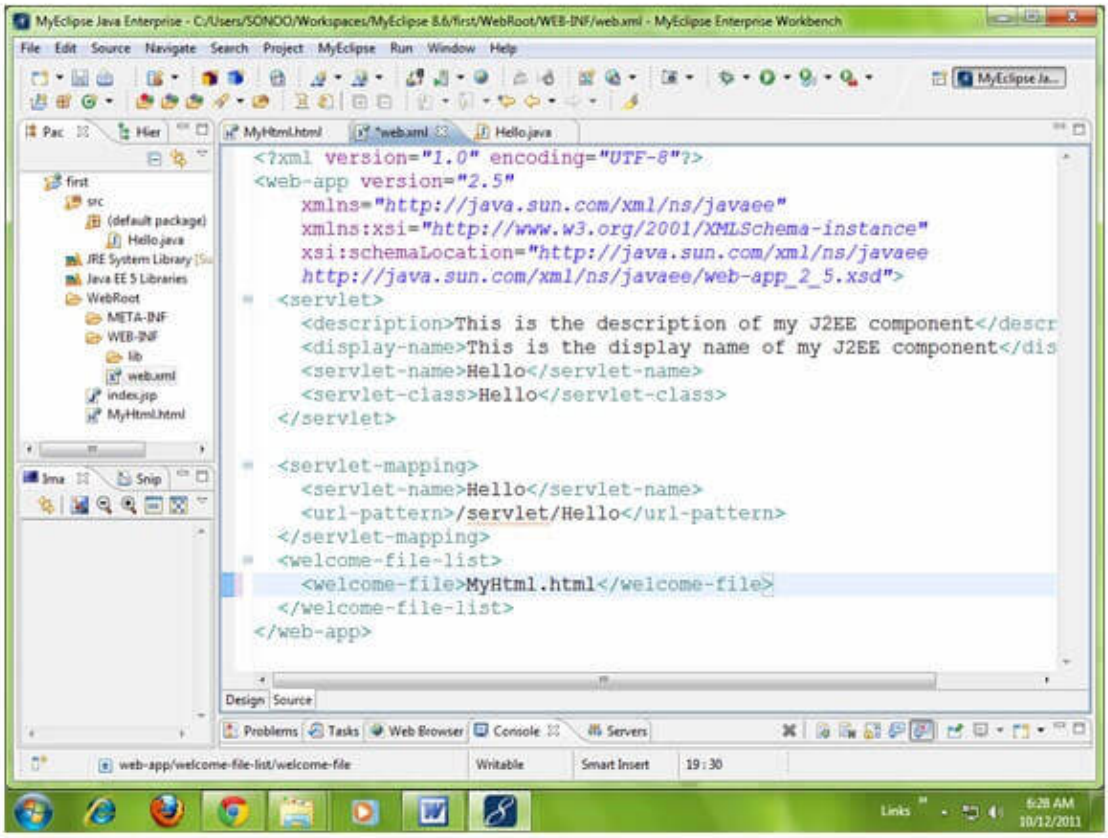
Now let's make the MyHtml.html file as the default page of our project. For this, open web.xml file and change the welcome file name as MyHtml.html in place of index.jsp.



Click on the source tab to see the source code.

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

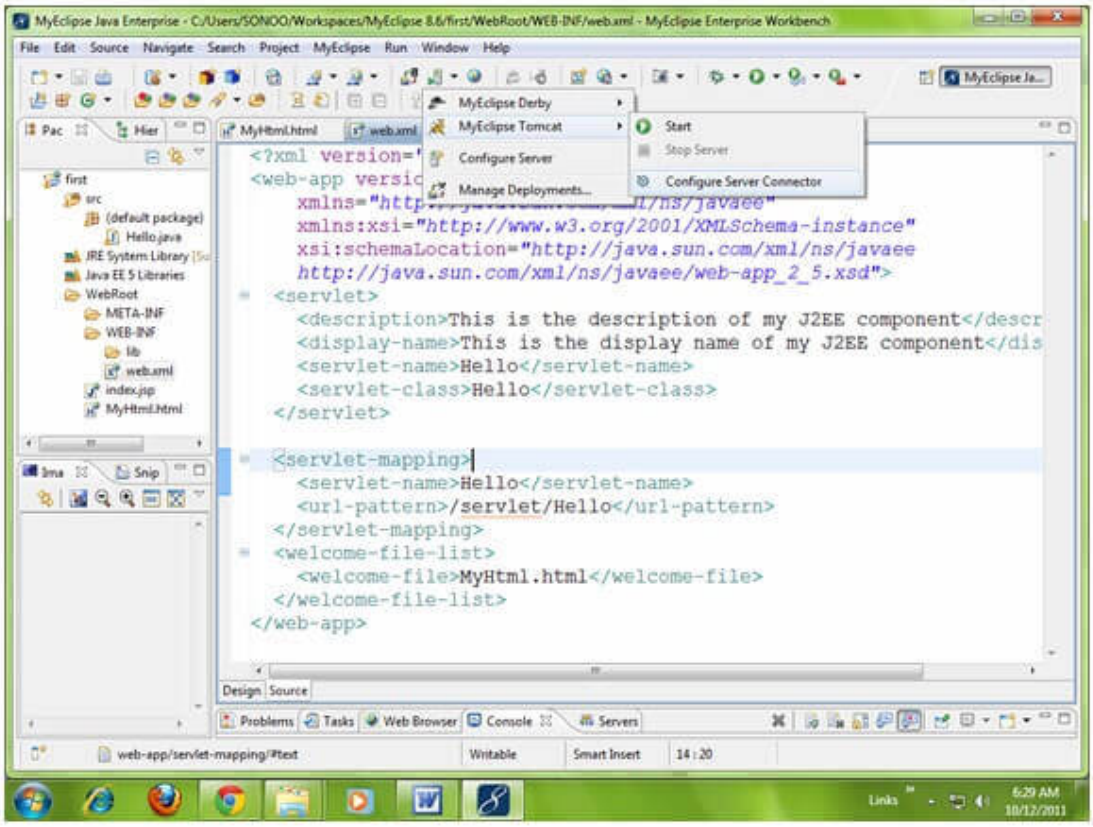




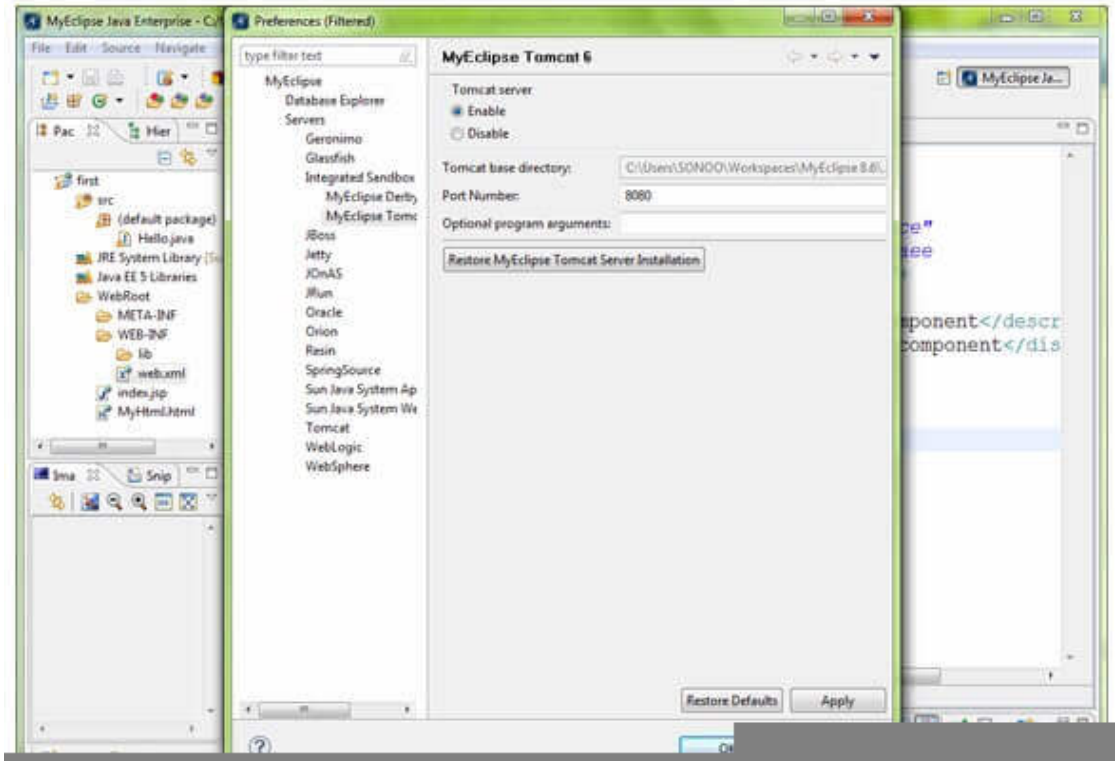
4) Start the server and deploy the project:

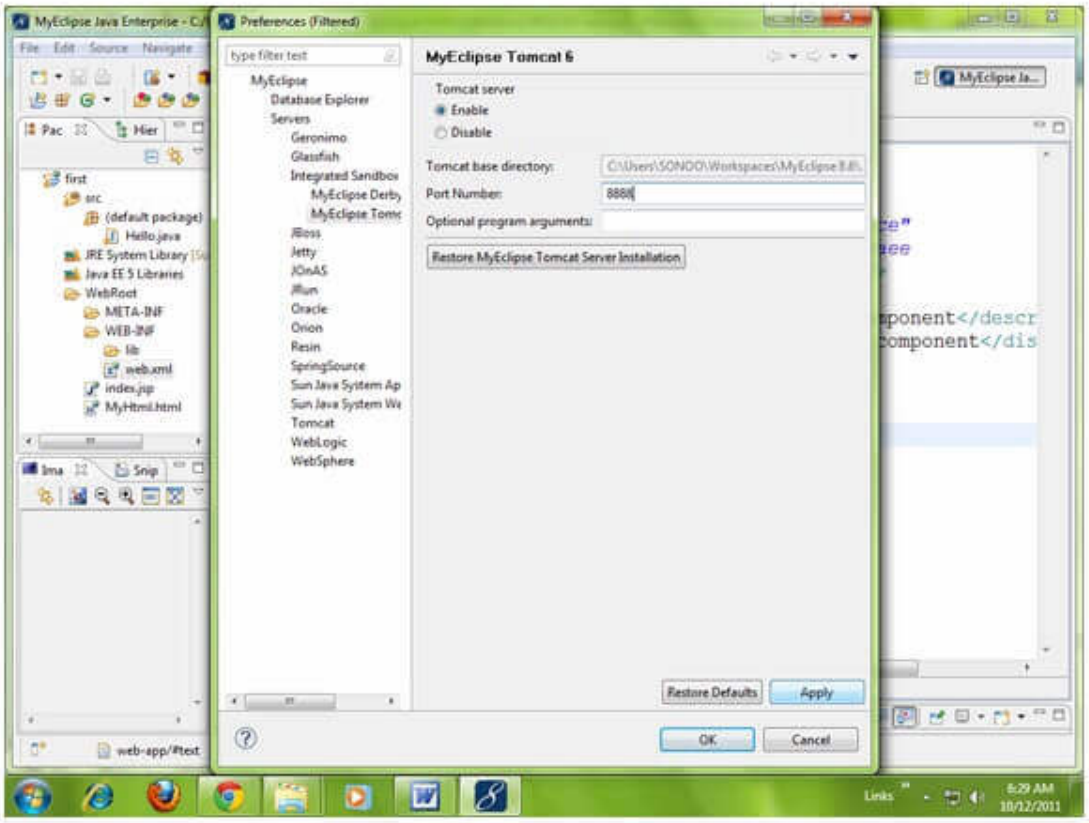
For starting the server and deploying the project in one step Right click on your project -> Run As -> MyEclipse server application.

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

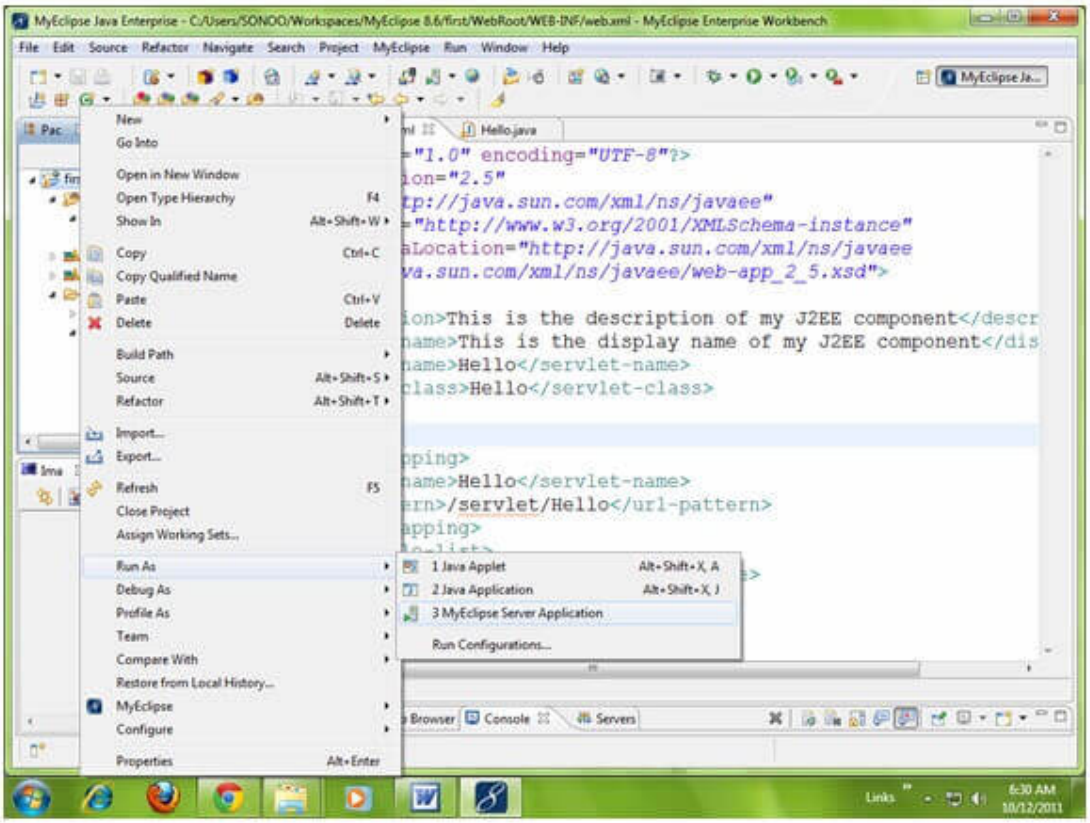


Now change the port number as 8888 in place of 8080 -> apply -> ok.

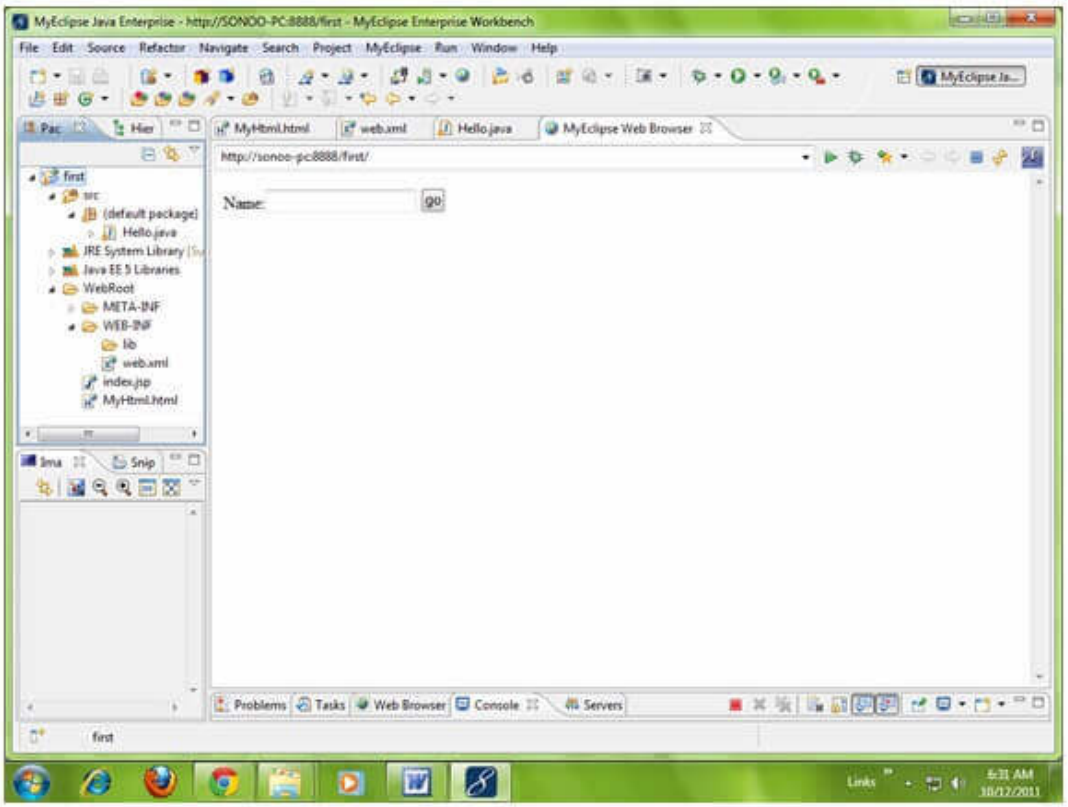

MyEclipse Java Enterprise - C:/
Preferences (Filtered)
type filter text
MyEclipse
Database Explorer
Servers
Geronimo
Glassfish
Integrated Sandbox
MyEclipse Derby
MyEclipse Tomc
JBoss
Jetty
JOnAS
JRun
Oracle
Orion
Resin
SpringSource
Sun Java System Ap
Sun Java System We
Tomcat
WebLogic
WebSphere
MyEclipse Tomcat 6
Tomcat server
Enable
Disable
Tomcat base directory:
C:\Users\SONOO\Workspaces\MyEclipse 8.6\...
Port Number:
8080
Optional program arguments:
Restore MyEclipse Tomcat Server Installation
Restore Defaults
Apply
MyEclipse Ja...
first
src
(default package)
Hello.java
JRE System Library
Java EE 5 Libraries
WebRoot
META-INF
WEB-INF
lib
web.xml
index.jsp
MyHtml.html
ponent</descr
component</dis

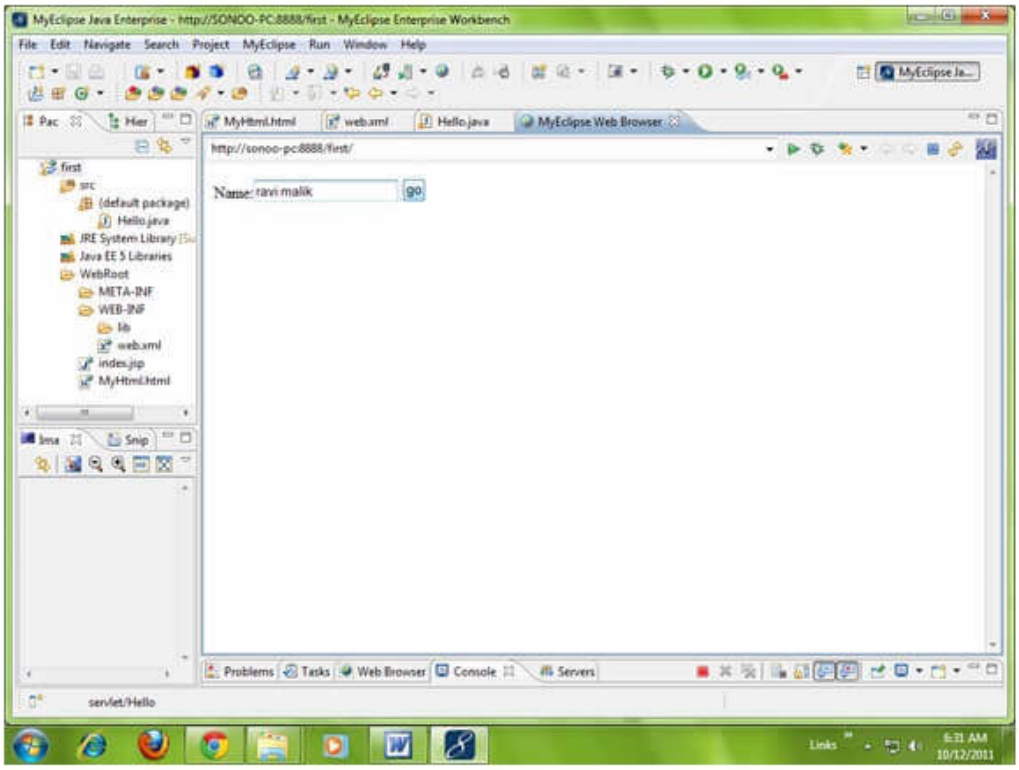

MyEclipse Java Enterprise - C:/
Preferences (Filtered)
type filter text
MyEclipse
Database Explorer
Servers
Geronimo
Glassfish
Integrated Sandbox
MyEclipse Derby
MyEclipse Tomc
JBoss
Jetty
JOnAS
JRun
Oracle
Orion
Resin
SpringSource
Sun Java System Ap
Sun Java System We
Tomcat
WebLogic
WebSphere
MyEclipse Tomcat 6
Tomcat server
Enable
Disable
Tomcat base directory:
C:\Users\SONOO\Workspaces\MyEclipse 8.6\...
Port Number:
8888
Optional program arguments:
Restore MyEclipse Tomcat Server Installation
Restore Defaults
Apply
OK
Cancel
MyEclipse Ja...
first
src
(default package)
Hello.java
JRE System Library
Java EE 5 Libraries
WebRoot
META-INF
WEB-INF
lib
web.xml
index.jsp
MyHtml.html
web-app/#test
ponent</descr
component</dis
Links
6:29 AM
10/12/2011

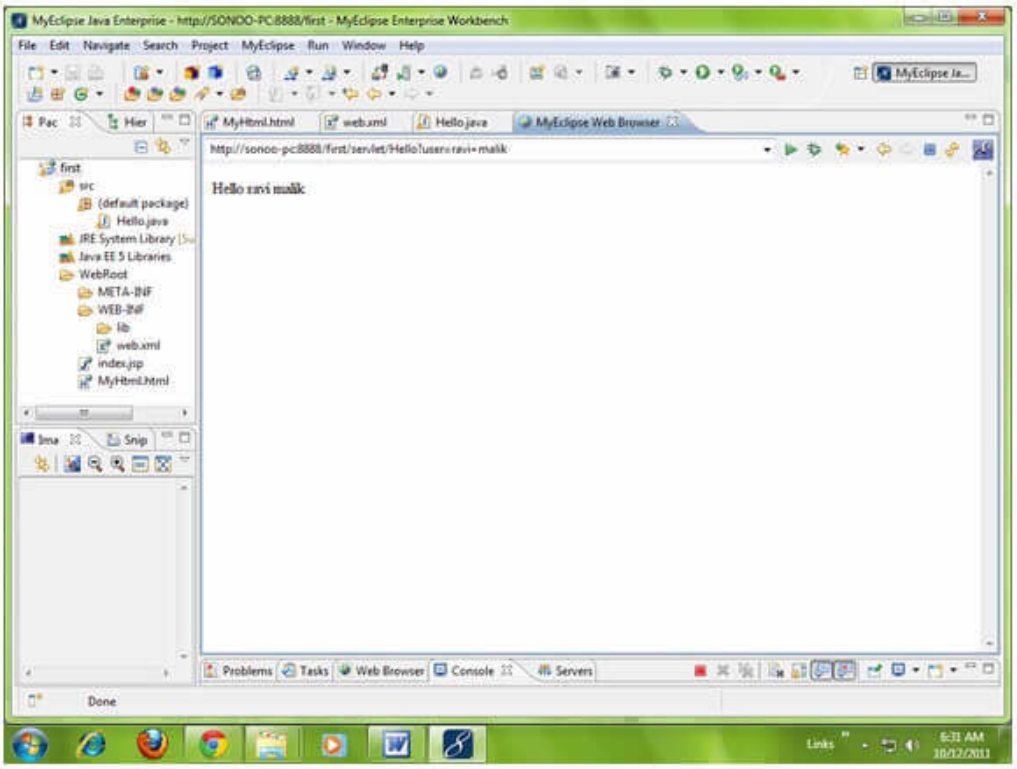
Now port number have been changed. For starting the server Right click on your project -> Run As -> MyEclipse server application.



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


MyEclipse Java Enterprise - http://SONOO-PC:8888/first - MyEclipse Enterprise Workbench
File Edit Source Refactor Navigate Search Project MyEclipse Run Window Help
MyHtml.html
web.xml
Hello.java
MyEclipse Web Browser
http://sonoo-pc:8888/first/
Name:
go
first
src
(default package)
Hello.java
JRE System Library
Java EE 5 Libraries
WebRoot
META-INF
WEB-INF
lib
web.xml
index.jsp
MyHtml.html
Problems
Tasks
Web Browser
Console
Servers
first


MyEclipse Java Enterprise - http://SONOO-PC:8888/first - MyEclipse Enterprise Workbench
File Edit Navigate Search Project MyEclipse Run Window Help
MyHtml.html
web.xml
Hello.java
MyEclipse Web Browser
http://sonoo-pc:8888/first/
Name: ravi malik
go
first
src
(default package)
Hello.java
JRE System Library
Java EE 5 Libraries
WebRoot
META-INF
WEB-INF
lib
web.xml
index.jsp
MyHtml.html
Problems
Tasks
Web Browser
Console
Servers
servlet/Hello



RequestDispatcher in Servlet

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

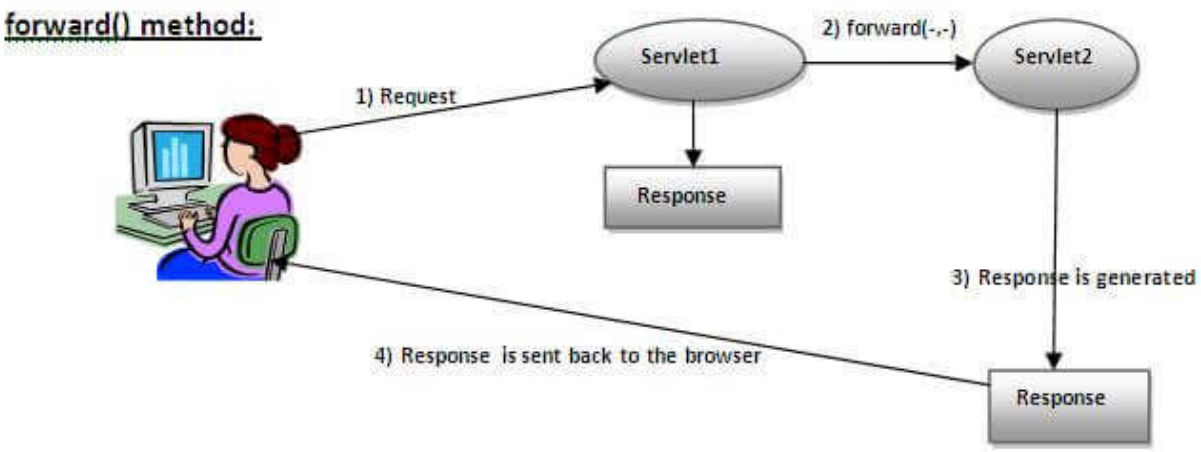
There are two methods defined in the RequestDispatcher interface.

Methods of RequestDispatcher interface

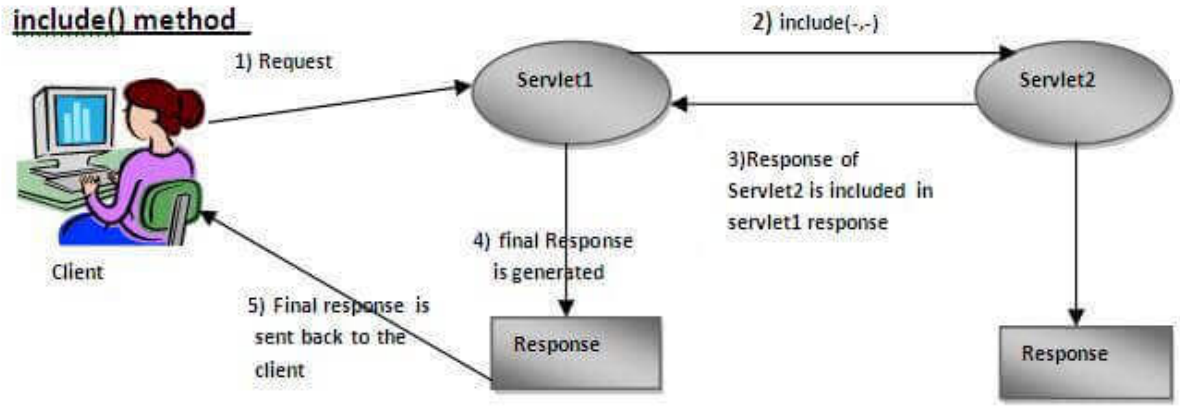
The RequestDispatcher interface provides two methods. They are:

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

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



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



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

How to get the object of RequestDispatcher

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

Syntax of getRequestDispatcher method

```
1. public RequestDispatcher getRequestDispatcher(String resou
   rce);
```

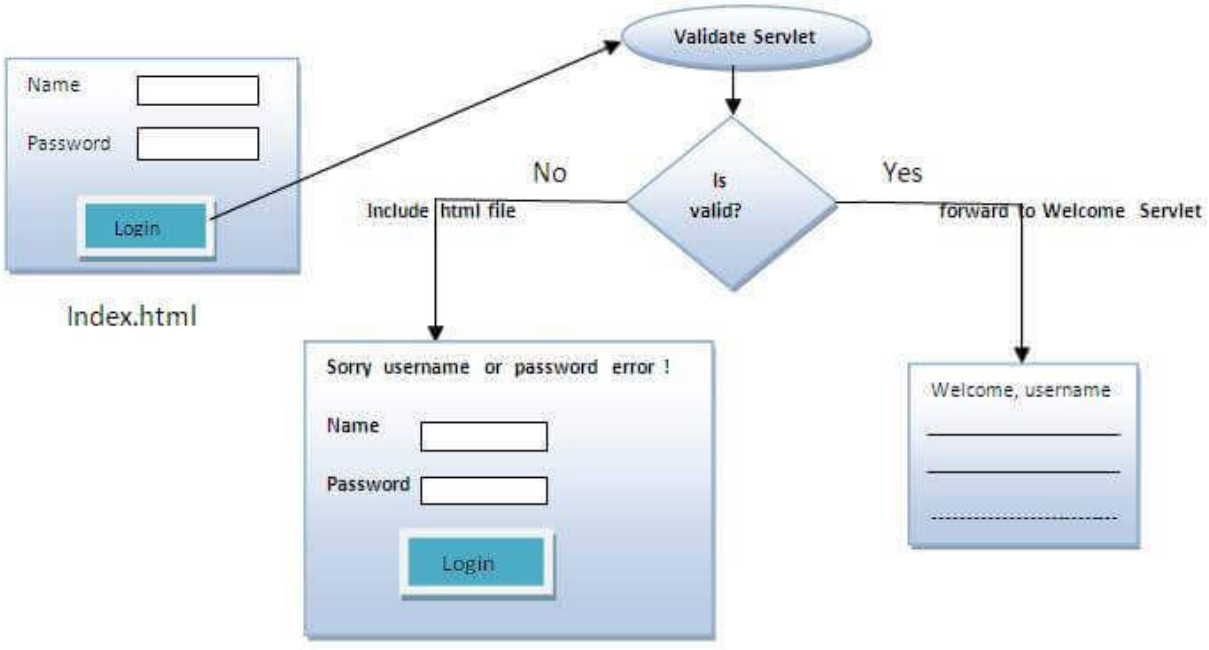
Example of using getRequestDispatcher method

```
1. RequestDispatcher  rd=request.getRequestDispatcher("servlet
   2");
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.



starting - index.html - Login.java - form validation - correct - forward method in RequestDispatcher - WelcomeServlet.java

starting - index.html - Login.java - form validation - Incorrect - include method in RequestDispatcher - index.html

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!");
```

```
23.         RequestDispatcher rd=request.getRequestDispatcher("/
    index.html");
24.         rd.include(request, response);
25.
26.         }
27.     }
28.
29. }
```

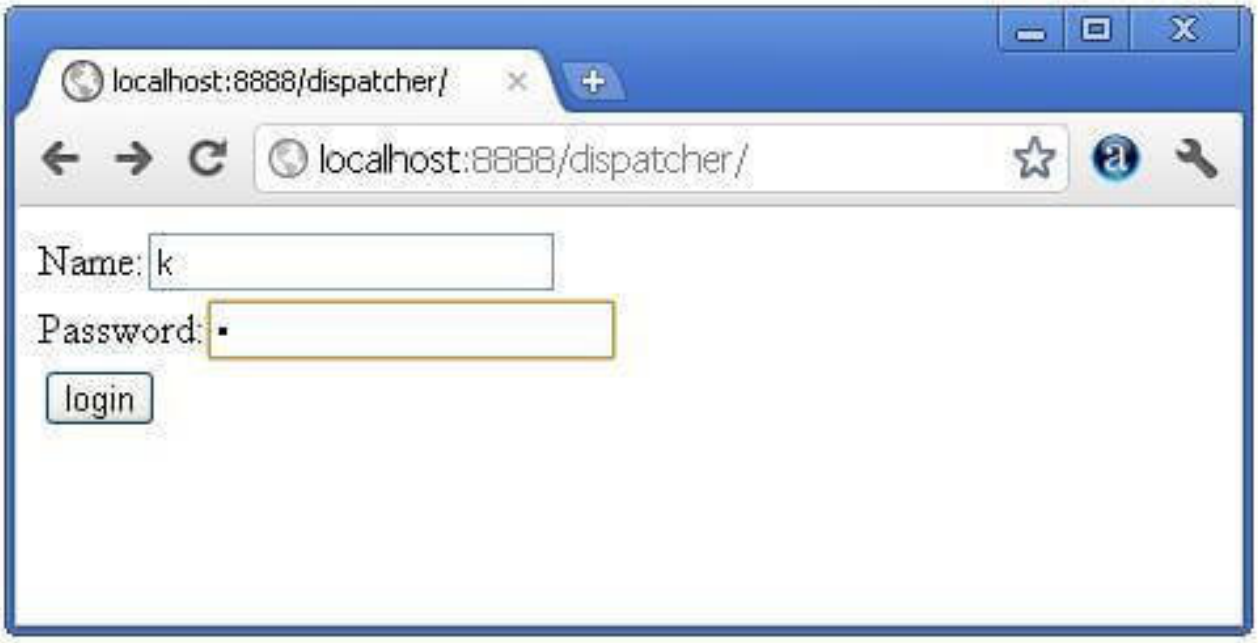
WelcomeServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.  public class WelcomeServlet extends HttpServlet {
6.
7.      public void doPost(HttpServletRequest request, HttpServle
    tResponse response)
8.          throws ServletException, IOException {
9.
10.     response.setContentType("text/html");
11.     PrintWriter out = response.getWriter();
12.
13.     String n=request.getParameter("userName");
14.     out.print("Welcome "+n);
15.     }
16.
17. }
```

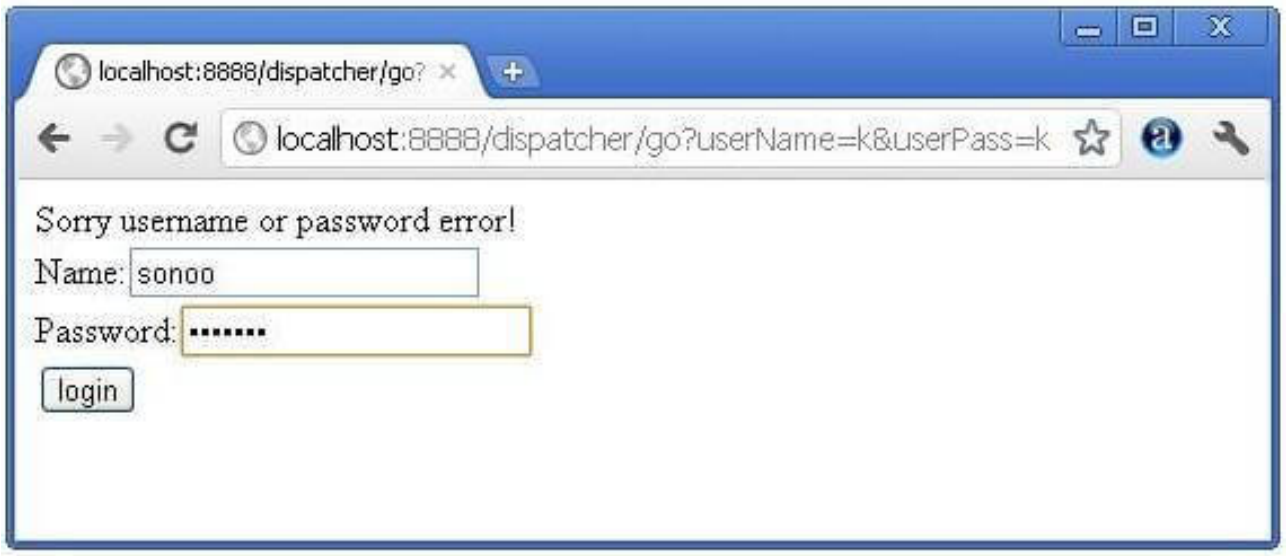
web.xml

```
1.  <web-app>
2.   <servlet>
3.     <servlet-name>Login</servlet-name>
4.     <servlet-class>Login</servlet-class>
5.    </servlet>
6.    <servlet>
7.     <servlet-name>WelcomeServlet</servlet-name>
```

```
8.     <servlet-class>WelcomeServlet</servlet-class>
9.   </servlet>
10.
11.
12.  <servlet-mapping>
13.    <servlet-name>Login</servlet-name>
14.    <url-pattern>/servlet1</url-pattern>
15.  </servlet-mapping>
16.  <servlet-mapping>
17.    <servlet-name>WelcomeServlet</servlet-name>
18.    <url-pattern>/servlet2</url-pattern>
19.  </servlet-mapping>
20.
21.  <welcome-file-list>
22.   <welcome-file>index.html</welcome-file>
23.  </welcome-file-list>
24. </web-app>
```


localhost:8888/dispatcher/
localhost:8888/dispatcher/
Name: k
Password:
login


localhost:8888/dispatcher/go?
localhost:8888/dispatcher/go?userName=k&userPass=k
Sorry username or password error!
Name:
Password:
login



SendRedirect in servlet

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

It accepts relative as well as absolute URL(google search - any weblink).

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.javatpoint.com");
```

Full example of sendRedirect method in servlet

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

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class DemoServlet extends HttpServlet{
6. public void doGet(HttpServletRequest req,HttpServletRespon
   se res)
7. throws ServletException,IOException
8. {
9. res.setContentType("text/html");
10. PrintWriter pw=res.getWriter();
11.
12. response.sendRedirect("http://www.google.com");
13.
14. pw.close();
15. }}
```

Creating custom google search using sendRedirect

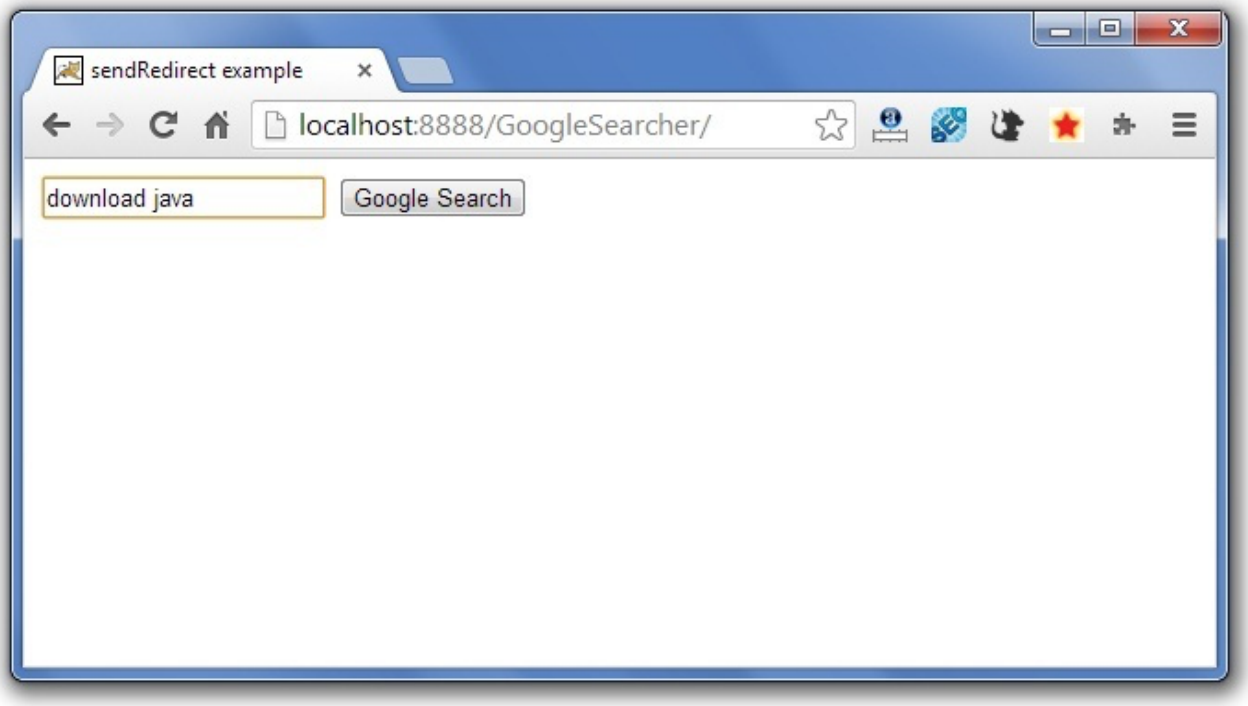
In this example, we are using sendRedirect method to send request to google server with the request data.

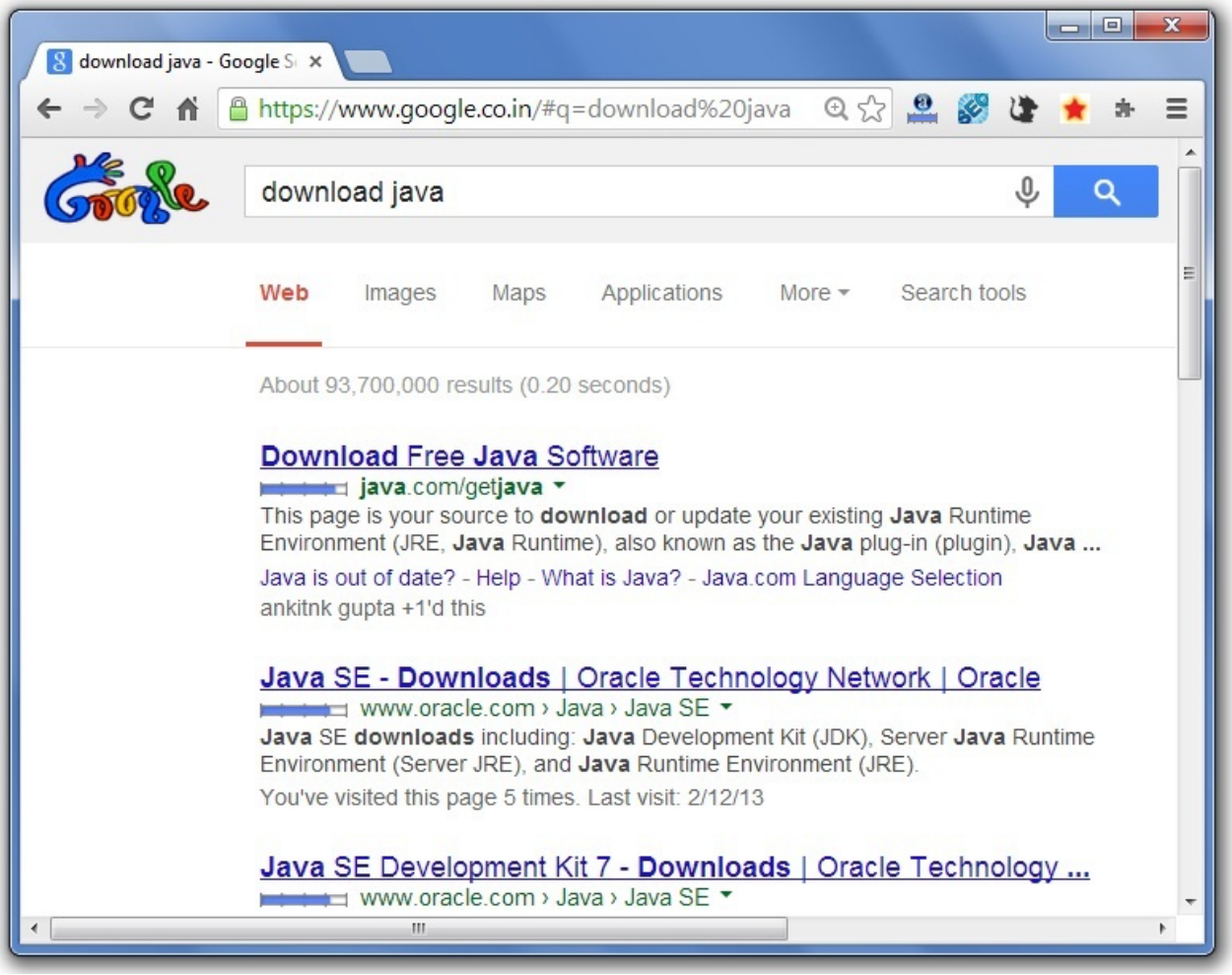
index.html

```
<!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, HttpSer
vletResponse response)
            throws ServletException, IOException {

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


sendRedirect example
localhost:8888/GoogleSearcher/
download java
Google Search


download java - Google S
https://www.google.co.in/#q=download%20java
download java
Web
Images
Maps
Applications
More
Search tools
About 93,700,000 results (0.20 seconds)
Download Free Java Software
java.com/getjava
This page is your source to download or update your existing Java Runtime Environment (JRE, Java Runtime), also known as the Java plug-in (plugin), Java ...
Java is out of date? - Help - What is Java? - Java.com Language Selection
ankitnk gupta +1'd this
Java SE - Downloads | Oracle Technology Network | Oracle
www.oracle.com › Java › Java SE
Java SE downloads including: Java Development Kit (JDK), Server Java Runtime Environment (Server JRE), and Java Runtime Environment (JRE).
You've visited this page 5 times. Last visit: 2/12/13
Java SE Development Kit 7 - Downloads | Oracle Technology ...
www.oracle.com › Java › Java SE

ServletConfig Interface

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

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

Advantage of ServletConfig

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

Methods of ServletConfig interface

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

How to get the object of ServletConfig

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

Syntax of getServletConfig() method

```
1. public ServletConfig getServletConfig();
```

Example of getServletConfig() method

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

Syntax to provide the initialization parameter for a servlet

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

```
1. <web-app>
2.   <servlet>
3.     ......
4.
5.     <init-param>
6.       <param-name>parametername</param-name>
7.       <param-value>parametervalue</param-value>
8.     </init-param>
9.     ......
10.  </servlet>
11. </web-app>
```

Example of ServletConfig to get initialization parameter

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

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
```

```
4.
5.  public class DemoServlet extends HttpServlet {
6.  public void doGet(HttpServletRequest request, HttpServletRe
    sponse response)
7.     throws ServletException, IOException {
8.
9.     response.setContentType("text/html");
10.    PrintWriter out = response.getWriter();
11.
12.    ServletConfig config=getServletConfig();
13.    String driver=config.getInitParameter("driver");
14.    out.print("Driver is: "+driver);
15.
16.    out.close();
17.    }
18.
19. }
```

web.xml

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>DemoServlet</servlet-name>
5.  <servlet-class>DemoServlet</servlet-class>
6.
7.  <init-param>
8.  <param-name>driver</param-name>
9.  <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-
    value>
10. </init-param>
11.
12. </servlet>
13.
14. <servlet-mapping>
15. <servlet-name>DemoServlet</servlet-name>
16. <url-pattern>/servlet1</url-pattern>
17. </servlet-mapping>
18.
19. </web-app>
```

Example of ServletConfig to get all the initialization parameters

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

DemoServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3. import java.util.Enumeration;
4.
5. import javax.servlet.ServletConfig;
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11.
12. public class DemoServlet extends HttpServlet {
13. public void doGet(HttpServletRequest request, HttpServletRe
    sponse response)
14.         throws ServletException, IOException {
15.
16.   response.setContentType("text/html");
17.   PrintWriter out = response.getWriter();
18.
19.   ServletConfig config=getServletConfig();
20.   Enumeration<String> e=config.getInitParameterNames();
21.
22.   String str="";
23.   while(e.hasMoreElements()){
24.   str=e.nextElement();
25.   out.print("<br>Name: "+str);
```

```
    out.print(" value: "+config.getInitParameter(str));
    }

    out.close();
}

}
```

web.xml

```
<web-app>

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

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

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

</servlet>

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

</web-app>
```

ServletContext Interface

Interface ServletContext. public interface ServletContext. **Defines a set of methods that a servlet uses to communicate with its servlet container**

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

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

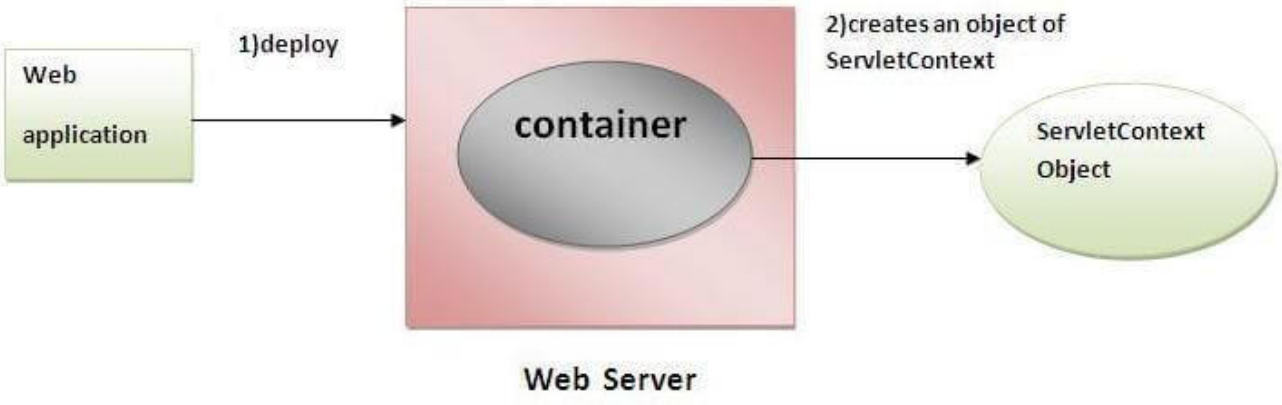
Advantage of ServletContext

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

Usage of ServletContext Interface

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

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



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.

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

How to get the object of ServletContext interface

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

Syntax of getServletContext() method

```
1. public ServletContext getServletContext()
```

Example of getServletContext() method

```
1. We can get the ServletContext object from ServletConfig obje
   ct
2. ServletContext  application=getServletConfig().getServletCont
   ext();

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

The ServletConfig parameters are specified for a particular servlet and are unknown to other servlets. It is used for intializing purposes.

The ServletContext parameters are specified for an entire application outside of any particular servlet and are available to all the servlets within that application. It is application scoped and thus globally accessible across the pages.

Syntax to provide the initialization parameter in Context scope

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

```
1. <web-app>
2.  ......
3.
4.   <context-param>
5.     <param-name>parametername</param-name>
6.     <param-value>parametervalue</param-value>
7.   </context-param>
```

8. .....
9. </web-app>

Example of ServletContext to get the initialization parameter

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

Difference between Context Init Parameters and Servlet Init Parameter

Context Init parameters	Servlet Init
Available to all servlets and JSPs that are part of web	Available to only servlet for which the <init-param>
Context Init parameters are initi <web-app> not within a specific elements	Initialized within the < for each specific serv
ServletContext object is used to get Context Init parameters	ServletConfig object is used to

Only one ServletContext object for entire web app	Each servlet has its own ServletConfig

DemoServlet.java

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


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

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

Getting the value of the initialization parameter and printing it

String driverName=context.getInitParameter("dname");
pw.println("driver name is="+driverName);

pw.close();

}}
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>ServletContext</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
```

```
7.
8. <context-param>
9. <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
11. </context-param>
12.
13. <servlet-mapping>
14. <servlet-name>ServletContext</servlet-name>
15. <url-pattern>/context</url-pattern>
16. </servlet-mapping>
17.
18. </web-app>
```

Example of ServletContext to get all the initialization parameters

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

DemoServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5.
6. public class DemoServlet extends HttpServlet{
7. public void doGet(HttpServletRequest req,HttpServletResponse res)
8. throws ServletException,IOException
9. {
10. res.setContentType("text/html");
11. PrintWriter out=res.getWriter();
12.
13. ServletContext context=getServletContext();
14. Enumeration<String> e=context.getInitParameterNames();
```

```
15.
16. String str="";
17. while(e.hasMoreElements()){
18.     str=e.nextElement();
19.     out.print("<br> "+context.getInitParameter(str));
20. }
21. }}
```

web.xml

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>sonoojaiswal</servlet-name>
5.  <servlet-class>DemoServlet</servlet-class>
6.  </servlet>
7.
8.  <context-param>
9.  <param-name>dname</param-name>
10. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-
    value>
11. </context-param>
12.
13. <context-param>
14. <param-name>username</param-name>
15. <param-value>system</param-value>
16. </context-param>
17.
18. <context-param>
19. <param-name>password</param-name>
20. <param-value>oracle</param-value>
21. </context-param>
22.
23. <servlet-mapping>
24. <servlet-name>sonoojaiswal</servlet-name>
25. <url-pattern>/context</url-pattern>
26. </servlet-mapping>
27.
28. </web-app>
```

Create the Servlet Context application - Registration form

Firstname :

Lastname:

Course:

Contact no:

Email id:

Submit

index.html

if course = Java = display Java Course details - Syllabus, duration, Cost. Place a tabular column in JavaServlet

Course	Syllabus	duration	Cost
Core Java	Threading, FileStreaming, Wrapper Class, Exception	2 months	15000
J2EE	Core java, JDBC, Servlet	3 months	30000
Java Full Stack	HTML, Core Java, JDBC. Servlet, Spring, Hibernate, SpringBoot, Angular	5 months	45000

if course = Finance = display Finance course details - FinanceServlet

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.

Project name - Servlet_Attribute

DemoServlet1.java

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

Create index.html
Login page
Username:
Password:

Create a LoginPage.java
if password = testing go to DemoServlet1.java
else goto index.html

web.xml write the welcome-file code and LoginPage servlet code

DemoServlet2.java

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

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

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

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

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

web.xml

```
<web-app>

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

<servlet-mapping>
```

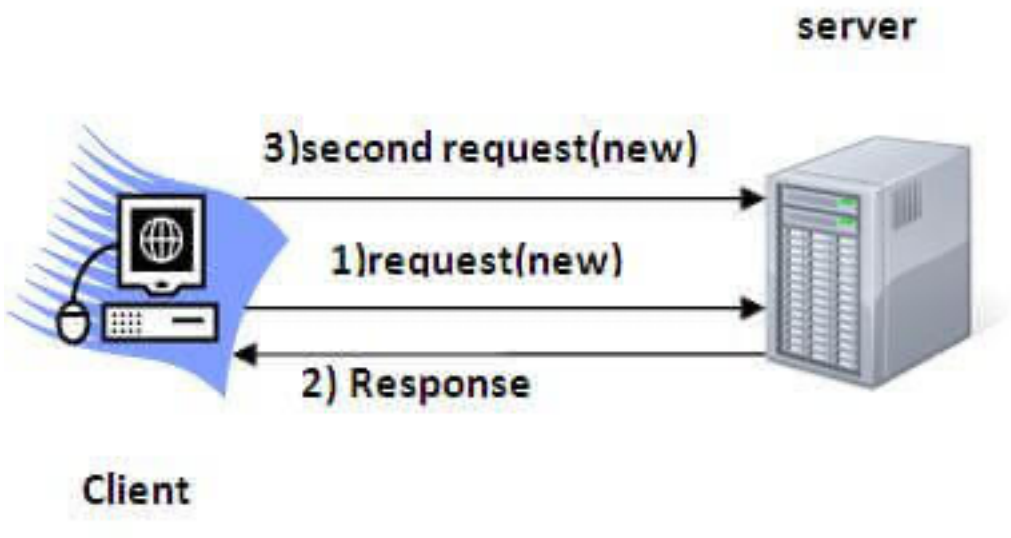
```
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

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

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

</web-app>
```

Session Tracking in Servlets

Session simply means a particular interval of time.

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

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server,



server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

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

Why use Session Tracking?

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

Session Tracking Techniques

There are four techniques used in Session tracking:

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

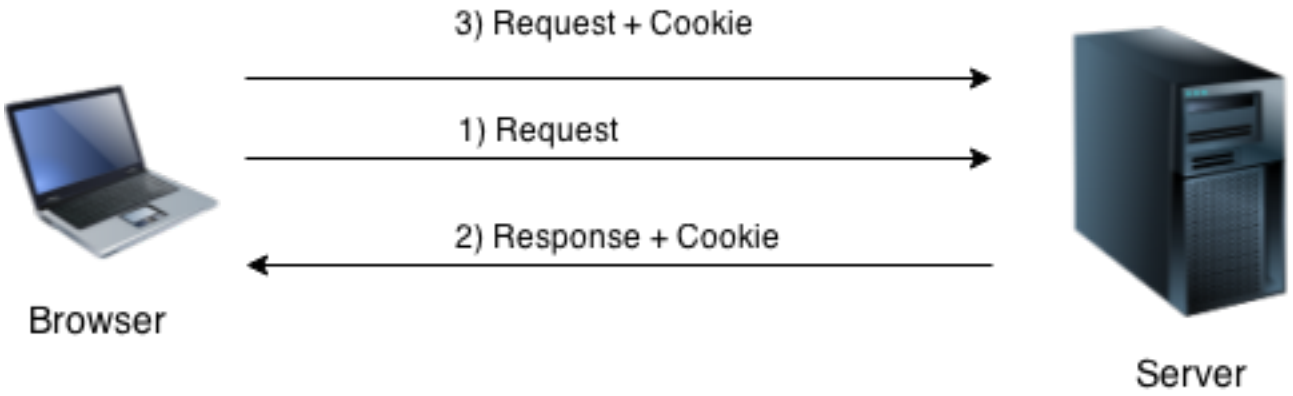
Cookies in Servlet

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

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

How Cookie works

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



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

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

Persistent cookie

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

Advantage of Cookies

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

Disadvantage of Cookies

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

Cookie class

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

Constructor of Cookie class

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

Useful Methods of Cookie class

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

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

public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

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

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

How to create Cookie?

Let's see the simple code to create cookie.

```
1. Cookie ck=new Cookie("user","sonoo jaiswal”);
2. //creating cookie object
3. 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.

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

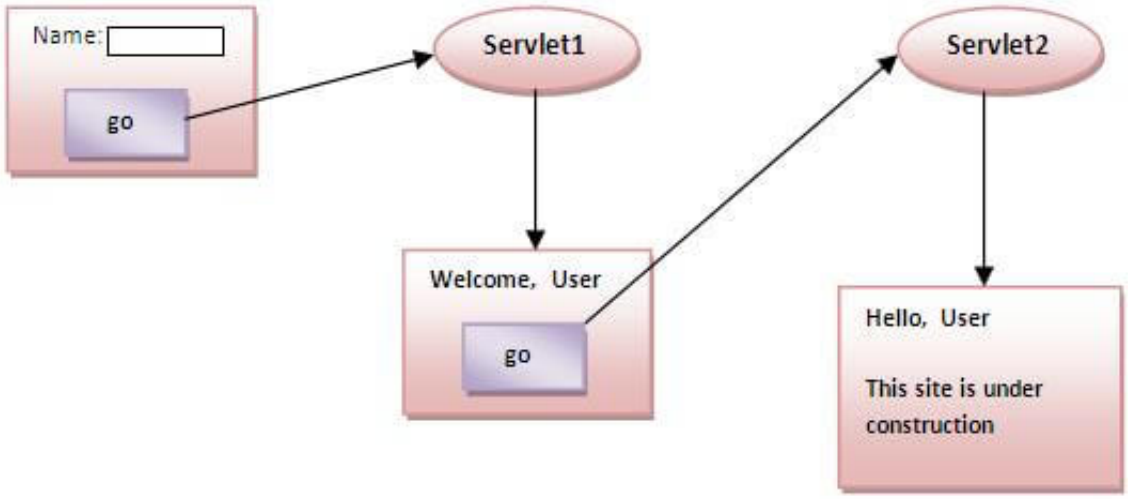
How to get Cookies?

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

```
1. Cookie ck[]=request.getCookies();
2. for(int i=0;i<ck.length;i++){
3.   out.print("<br>"+ck[i].getName()+"   "+ck[i].getValue());//
   printing name and value of cookie
4. }
```

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.



Servlet_Cookies - Project name

index.html

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

FirstServlet.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.  import javax.servlet.http.*;
4.
5.
6.  public class FirstServlet extends HttpServlet {
7.
8.    public void doPost(HttpServletRequest request, HttpServlet
    Response response){
9.      try{
10.
11.     response.setContentType("text/html");
12.     PrintWriter out = response.getWriter();
13.
14.     String n=request.getParameter("userName");
15.     out.print("Welcome "+n);
16.
17.           Cookie ck=new Cookie("uname",n);//
    creating cookie object
18.     response.addCookie(ck);//adding cookie in the response
19.
20.     //creating submit button
21.     out.print("<form action='servlet2'>");
22.     out.print("<input type='submit' value='go'>");
23.     out.print("</form>");
24.
25.     out.close();
26.
27.       }catch(Exception e){System.out.println(e);}
28.   }
29. }
```

SecondServlet.java

```
1. import java.io.*;
2. import javax.servlet.*;
3. import javax.servlet.http.*;
4.
5. public class SecondServlet extends HttpServlet {
6.
7. public void doPost(HttpServletRequest request, HttpServletR
   esponse response){
8.    try{
9.
10.   response.setContentType("text/html");
11.   PrintWriter out = response.getWriter();
12.
13.   Cookie ck[]=request.getCookies();
14.   out.print("Hello "+ck[0].getValue());
15.
16.   out.close();
17.
18.       }catch(Exception e){System.out.println(e);}
19.   }
20.
21.
22. }
```

web.xml

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12.
13. <servlet>
```

```
14. <servlet-name>s2</servlet-name>
15. <servlet-class>SecondServlet</servlet-class>
16. </servlet>
17.
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
22.
23. </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 example, we are creating 3 links: login, logout and profile. User can't go to profile page until he/she is logged in. If user is logged out, he need to login again to visit profile.

In this application, we have created following files.

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

Project name: Servlet_Login_Logout

File:index.html

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

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

</body>
</html>
```

File: link.html

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

File: login.html

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

File: LoginServlet.java

```
package com.javatpoint;

import java.io.IOException;
```

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

        request.getRequestDispatcher("link.html").include(request, res
ponse);

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

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

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

        out.close();
    }

}
```

File: LogoutServlet.java

```
package com.javatpoint;
```

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


         request.getRequestDispatcher("link.html").include(request, res
ponse);

        Cookie ck=new Cookie("name","");
        ck.setMaxAge(0);                //Delete the cookie
        response.addCookie(ck);

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

File: ProfileServlet.java

```
package com.javatpoint;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ProfileServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletR
esponse response)
                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();

         request.getRequestDispatcher("link.html").include(request, res
ponse);

        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"
x m l n s = " h t t p : / / j a v a . s u n . c o m / x m l / n s /
javaee" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
h t t p : / / j a v a . s u n . c o m / x m l / n s / j a v a e e / w e b -
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>
```

2) Hidden Form Field

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

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

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

```
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

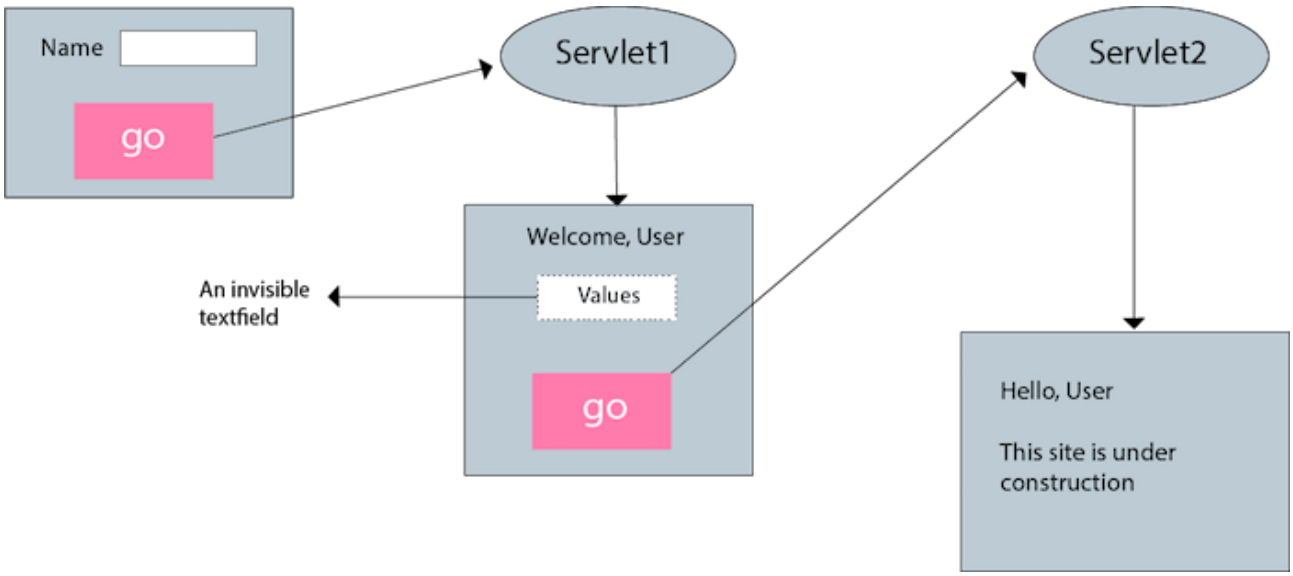
1. It will always work whether cookie is disabled or not.

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, HttpServletRespon
se 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, HttpServletRespon
se response)
        try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

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

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

web.xml

```
<web-app>

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

<servlet-mapping>
<servlet-name>s1</servlet-name>
```

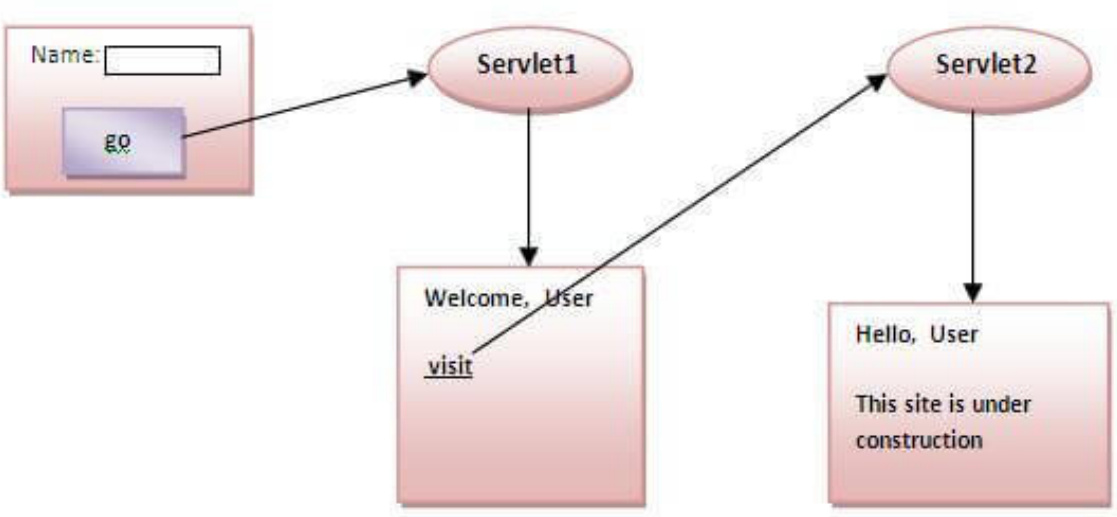
```
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

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

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

</web-app>
```

3)URL Rewriting

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

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

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



Advantage of URL Rewriting

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

Disadvantage of URL Rewriting

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

Example of using URL Rewriting

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

Project_name : URL_Rewriting

index.html

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

FirstServlet.java

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

        //getting value from the query string
        String n=request.getParameter("uname");
        out.print("Hello "+n);
        out.close();
       }catch(Exception e){System.out.println(e);}
    }
  }
```

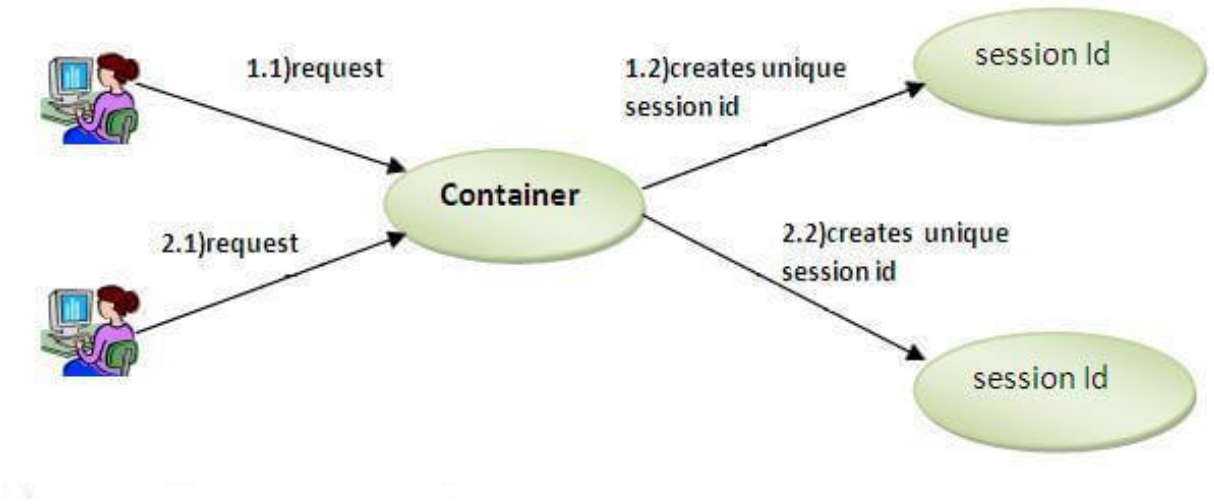
web.xml

```
<web-app>

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

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

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

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

</web-app>
```

4) HttpSession interface

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

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



How to get the HttpSession object ?

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

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

Commonly used methods of HttpSession interface

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

Example of using HttpSession

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

Cookies - Store the information on the client side

Session - Store the information on the Server side

HTTPSession - Project name

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, HttpServletRespon
se 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, HttpServletRespon
se response)
        try{

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

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

        out.close();

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

  }
```

web.xml

```
<web-app>

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

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

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

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

</web-app>
```

Servlet HttpSession Login and Logout Example

We can bind the objects on HttpSession instance and get the objects by using setAttribute and getAttribute methods.

In the previous page, we have learnt about what is HttpSession, How to store and get data from session object etc.

Here, we are going to create a real world login and logout application without using database code. We are assuming that password is admin123.

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

In this application, we have created following files.\

Servlet HttpSession Login and Logout Example

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

```
HttpSession session=request.getSession();
        session.setAttribute("name",name);
```

Project Name: Servlet_HTTPSession_Login_Logout

File: index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Servlet Login Example</title>
</head>
<body>
<h1>Login App using HttpSession</h1>
<a href="login.html">Login</a>|
<a href="LogoutServlet">Logout</a>|
```

```
<a href="ProfileServlet">Profile</a>
</body>
</html>
```

File: link.html

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

File: login.html

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

File: LoginServlet.java

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

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServlet
Response response)
                throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
         request.getRequestDispatcher("link.html").include(request, res
ponse);
```

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

        if(password.equals("admin123")){
        out.print("Welcome, "+name);
        HttpSession session=request.getSession();
        session.setAttribute("name",name);
        }
        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.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class LogoutServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServle
tResponse response)
                            throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out=response.getWriter();

            request.getRequestDispatcher("link.html").include(request, r
esponse);

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

```
            session.invalidate();

            out.print("You are successfully logged out!");

            out.close();
    }
}
```

File: ProfileServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class ProfileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletR
esponse response)
                    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
         request.getRequestDispatcher("link.html").include(request, res
ponse);

        HttpSession session=request.getSession(false);
        if(session!=null){
        String name=(String)session.getAttribute("name");

        out.print("Hello, "+name+" Welcome to Profile");
        }
        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>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>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>LogoutServlet</servlet-class>
  </servlet>
```

```
  <servlet-mapping>
    <servlet-name>LogoutServlet</servlet-name>
    <url-pattern>/LogoutServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Event and Listener in Servlet

Events are basically occurrence of something. Changing the state of an object is known as an event.

We can perform some important tasks at the occurrence of these exceptions, such as counting total and current logged-in users, creating tables of the database at time of deploying the project, creating database connection object etc.

There are many Event classes and Listener interfaces in the javax.servlet and javax.servlet.http packages.

Event classes

The event classes are as follows:

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

Event interfaces

The event interfaces are as follows:

1. ServletRequestListener
2. ServletRequestAttributeListener

3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

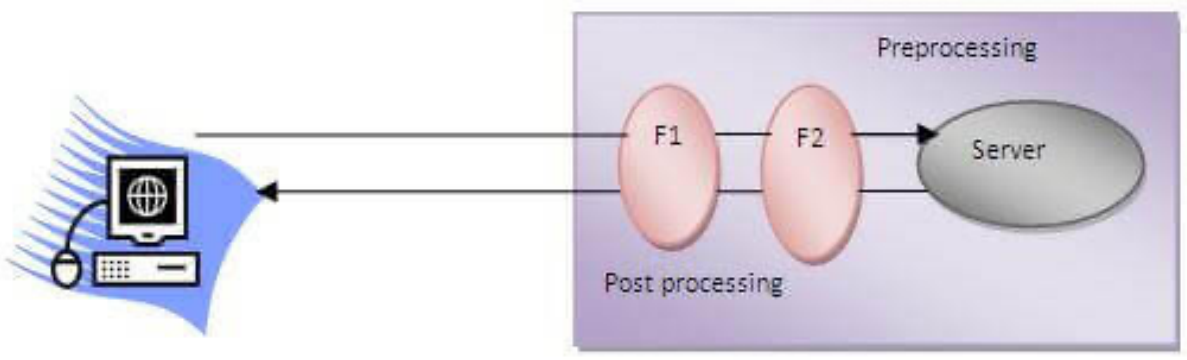
Servlet Filter

A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

So maintenance cost will be less.



Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

Advantage of Filter

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

Filter API

Like servlet filter have its own API. The javax.servlet package contains the three interfaces of Filter API.

1. Filter
2. FilterChain
3. FilterConfig

1) Filter interface

For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

Method	Description
public void init(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
public void doFilter(HttpServletRequest request,HttpServletResp	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped.It is used to perform filtering tasks

public void destroy()	This is invoked only once when filter is taken out of the service.

2) FilterChain interface

The object of FilterChain is responsible to invoke the next filter or resource in the chain.This object is passed in the doFilter method of Filter interface.The FilterChain interface contains only one method:

1. **public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

How to define Filter

We can define filter same as servlet. Let's see the elements of filter and filter-mapping.

```
<web-app>

<filter>
<filter-name>...</filter-name>
<filter-class>...</filter-class>
</filter>

<filter-mapping>
<filter-name>...</filter-name>
<url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

For mapping filter we can use, either url-pattern or servlet-name. The url-pattern elements has an advantage over servlet-name element i.e. it can be applied on servlet, JSP or HTML.

Simple Example of Filter

In this example, we are simply displaying information that filter is invoked automatically after the post processing of the request.

index.html

```
1. <a href="servlet1">click here</a>
```

MyFilter.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.*;
5.
6. public class MyFilter implements Filter{
7.
8. public void init(FilterConfig arg0) throws ServletException {}
9.
10. public void doFilter(ServletRequest req, ServletResponse res
    p,
11.     FilterChain chain) throws IOException, ServletException {
12.
13.     PrintWriter out=resp.getWriter();
14.     out.print("filter is invoked before");
15.
16.     chain.doFilter(req, resp);//sends request to next resource
17.
18.     out.print("filter is invoked after");
19.     }
20.     public void destroy() {}
21. }
```

HelloServlet.java

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

```
3.
4.  import javax.servlet.ServletException;
5.  import javax.servlet.http.*;
6.
7.  public class HelloServlet extends HttpServlet {
8.      public void doGet(HttpServletRequest request, HttpServlet
    Response response)
9.              throws ServletException, IOException {
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         out.print("<br>welcome to servlet<br>");
15.
16.     }
17.
18. }
```

web.xml

For defining the filter, filter element of web-app must be defined just like servlet.

```
1.  <web-app>
2.
3.  <servlet>
4.  <servlet-name>s1</servlet-name>
5.  <servlet-class>HelloServlet</servlet-class>
6.  </servlet>
7.
8.  <servlet-mapping>
9.  <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
12.
13. <filter>
14. <filter-name>f1</filter-name>
15. <filter-class>MyFilter</filter-class>
16. </filter>
17.
18. <filter-mapping>
19. <filter-name>f1</filter-name>
```

```
<url-pattern>/servlet1</url-pattern>
</filter-mapping>


</web-app>
```

Authentication Filter

We can perform authentication in filter. Here, we are going to check to password given by the user in filter class, if given password is admin, it will forward the request to the WelcomeAdmin servlet otherwise it will display error message.

Example of authenticating user using filter

Let's see the simple example of authenticating user using filter.

Here, we have created 4 files:

- index.html
- MyFilter.java
- AdminServlet.java
- web.xml

index.html

```
<form action="servlet1">
Name:<input type="text" name="name"/><br/>
Password:<input  type="password"  name="password"/><br/>

<input type="submit" value="login">

</form>
```

MyFilter.java

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

```
3. import javax.servlet.*;
4.
5. public class MyFilter implements Filter{
6.
7. public void init(FilterConfig arg0) throws ServletException {}
8.
9. public void doFilter(ServletRequest req, ServletResponse res
   p,
10.         FilterChain chain) throws IOException, ServletException
   {
11.
12.    PrintWriter out=resp.getWriter();
13.
14.    String password=req.getParameter("password");
15.    if(password.equals("admin")){
16.    chain.doFilter(req, resp);//sends request to next resource
17.    }
18.    else{
19.    out.print("username or password error!");
20.     RequestDispatcher rd=req.getRequestDispatcher("index.ht
   ml");
21.    rd.include(req, resp);
22.    }
23.
24. }
25.    public void destroy() {}
26.
27. }
```

AdminServlet.java

```
1. import java.io.IOException;
2. import java.io.PrintWriter;
3.
4. import javax.servlet.ServletException;
5. import javax.servlet.http.*;
6.
7. public class AdminServlet extends HttpServlet {
```

```
8.  public void doGet(HttpServletRequest request, HttpServletRe
    sponse response)
9.          throws ServletException, IOException {
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         out.print("welcome ADMIN");
15.         out.close();
16.     }
17. }
```

web.xml

```
1.  <web-app>
2.   <servlet>
3.     <servlet-name>AdminServlet</servlet-name>
4.     <servlet-class>AdminServlet</servlet-class>
5.   </servlet>
6.
7.   <servlet-mapping>
8.     <servlet-name>AdminServlet</servlet-name>
9.     <url-pattern>/servlet1</url-pattern>
10.  </servlet-mapping>
11.
12. <filter>
13.  <filter-name>f1</filter-name>
14.  <filter-class>MyFilter</filter-class>
15.  </filter>
16.  <filter-mapping>
17.  <filter-name>f1</filter-name>
18.  <url-pattern>/servlet1</url-pattern>
19.  </filter-mapping>
20.
21. </web-app>
```

FilterConfig

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

Methods of FilterConfig interface

There are following 4 methods in the FilterConfig interface.

1. **public void init(FilterConfig config):** init() method is invoked only once it is used to initialize the filter.
2. **public String getInitParameter(String parameterName):** Returns the parameter value for the specified parameter name.
3. **public java.util.Enumeration getInitParameterNames():** Returns an enumeration containing all the parameter names.
4. **public ServletContext getServletContext():** Returns the ServletContext object.

Example of FilterConfig

In this example, if you change the param-value to no, request will be forwarded to the servlet otherwise filter will create the response with the message: this page is underprocessing. Let's see the simple example of FilterConfig. Here, we have created 4 files:

- index.html
- MyFilter.java
- HelloServlet.java
- web.xml

index.html

```
1. <a href="servlet1">click here</a>
```

MyFilter.java

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

```
3.
4.  import javax.servlet.*;
5.
6.  public class MyFilter implements Filter{
7.  FilterConfig config;
8.
9.  public void init(FilterConfig config) throws ServletException {

10.     this.config=config;
11. }
12.
13. public void doFilter(ServletRequest req, ServletResponse res
    p,
14.     FilterChain chain) throws IOException, ServletException {
15.
16.     PrintWriter out=resp.getWriter();
17.
18.     String s=config.getInitParameter("construction");
19.
20.     if(s.equals("yes")){
21.         out.print("This page is under construction");
22.     }
23.     else{
24.                                 chain.doFilter(req, resp);//
    sends request to next resource
25.     }
26.
27. }
28. public void destroy() {}
29. }
```

HelloServlet.java

```
1.  import java.io.IOException;
2.  import java.io.PrintWriter;
3.
4.  import javax.servlet.ServletException;
5.  import javax.servlet.http.*;
6.
```

```
7.  public class HelloServlet extends HttpServlet {
8.  public void doGet(HttpServletRequest request, HttpServletRe
    sponse response)
9.          throws ServletException, IOException {
10.
11.         response.setContentType("text/html");
12.         PrintWriter out = response.getWriter();
13.
14.         out.print("<br>welcome to servlet<br>");
15.
16.     }
17.
18. }
```

web.xml

```
1.  <web-app>
2.
3.   <servlet>
4.     <servlet-name>HelloServlet</servlet-name>
5.     <servlet-class>HelloServlet</servlet-class>
6.    </servlet>
7.
8.    <servlet-mapping>
9.     <servlet-name>HelloServlet</servlet-name>
10.    <url-pattern>/servlet1</url-pattern>
11.   </servlet-mapping>
12.
13.   <filter>
14.   <filter-name>f1</filter-name>
15.   <filter-class>MyFilter</filter-class>
16.   <init-param>
17.   <param-name>construction</param-name>
18.   <param-value>no</param-value>
19.   </init-param>
20.   </filter>
21.   <filter-mapping>
22.   <filter-name>f1</filter-name>
23.   <url-pattern>/servlet1</url-pattern>
24.   </filter-mapping>
```

```
25.
26.
27. </web-app>
```

Useful Filter Examples

There is given some useful examples of filter.

Example of sending response by filter only

MyFilter.java

```
1.  import java.io.*;
2.  import javax.servlet.*;
3.
4.  public class MyFilter implements Filter{
5.      public void init(FilterConfig arg0) throws ServletException {
    }
6.
7.      public void doFilter(ServletRequest req, ServletResponse r
    es,
8.              FilterChain chain) throws IOException, ServletExcepti
    on {
9.
10.         PrintWriter out=res.getWriter();
11.
12.         out.print("<br/>this site is underconstruction..");
13.         out.close();
14.
15.     }
16.     public void destroy() {}
17. }
```

Example of counting number of visitors for a single page

MyFilter.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class MyFilter implements Filter{
5.     static int count=0;
6.     public void init(FilterConfig arg0) throws ServletException {
   }
7.
8.      public void doFilter(ServletRequest req, ServletResponse r
   es,
9.              FilterChain chain) throws IOException, ServletExcepti
   on {
10.
11.         PrintWriter out=res.getWriter();
12.         chain.doFilter(request,response);
13.
14.         out.print("<br/>Total visitors "+(++count));
15.         out.close();
16.
17.     }
18.     public void destroy() {}
19. }
```

Example of checking total response time in filter

MyFilter.java

```
1. import java.io.*;
2. import javax.servlet.*;
3.
4. public class MyFilter implements Filter{
5.     static int count=0;
6.     public void init(FilterConfig arg0) throws ServletException {
   }
7.
8.      public void doFilter(ServletRequest req, ServletResponse r
   es,
```

```
9.              FilterChain chain) throws IOException, ServletExcepti
    on {
10.
11.         PrintWriter out=res.getWriter();
12.         long before=System.currentTimeMillis();
13.
14.         chain.doFilter(request,response);
15.
16.         long after=System.currentTimeMillis();
17.                 out.print("<br/>Total response time "+(after-before)
    +" miliseconds");
18.         out.close();
19.
20.     }
21.     public void destroy() {}
22. }
```