OROMIA STATE UNIVERSITY
COLLEGE OF SCINCE AND TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

CHAPTER THREE
SERVLETS AND JAVA SERVER PAGES

PART II

Compiled by :Yigermal Semahegn(M.Tech)

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

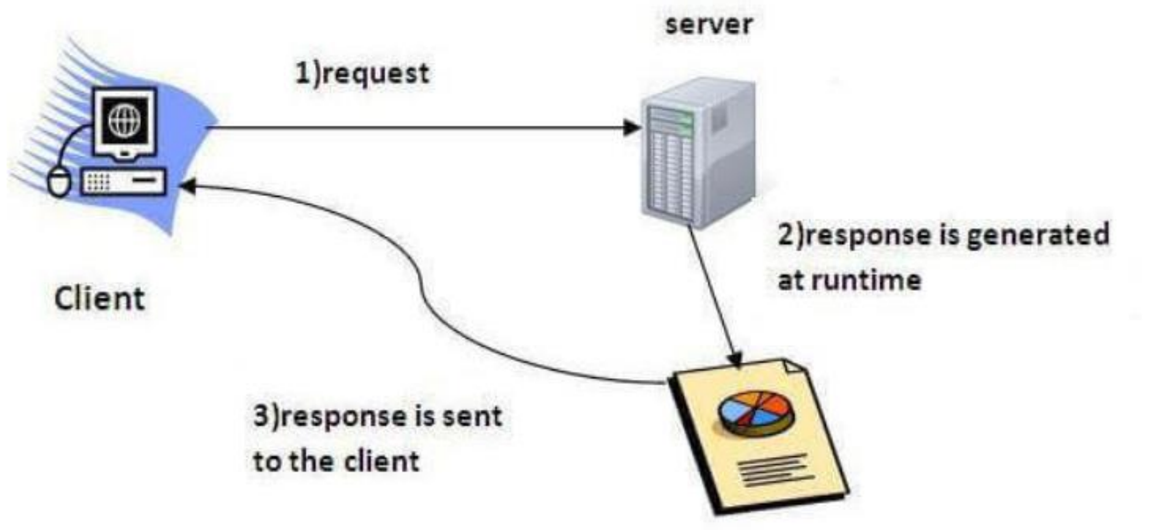
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

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.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

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

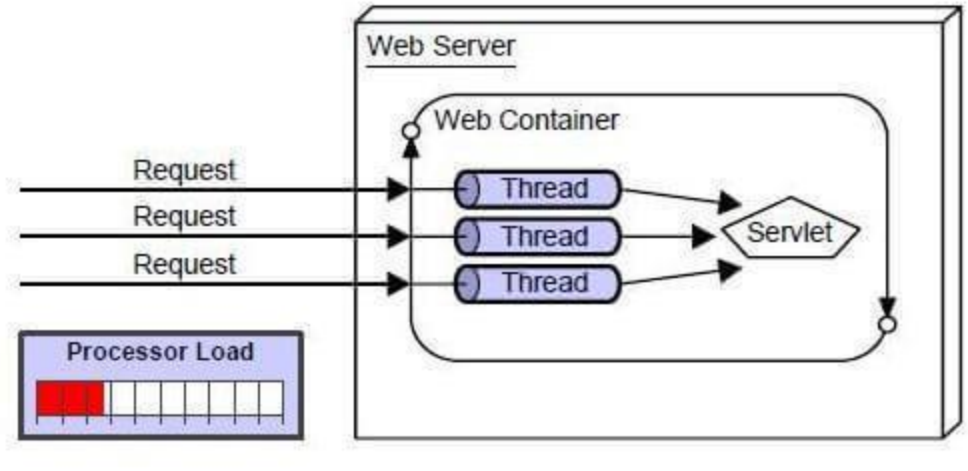
There are many problems in CGI technology:

If the number of clients increases, it takes more time for sending the response.

For each request, it starts a process, and the web server is limited to start processes.

It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet



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

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

Web Terminology

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

Website

Website is a collection of related web pages that may contain text, images, audio and video. The first page of a website is called home page. Each website has specific internet address (URL) that you need to enter in your browser to access a website.

Website is hosted on one or more servers and can be accessed by visiting its homepage using a computer network. A website is managed by its owner that can be an individual, company or an organization.

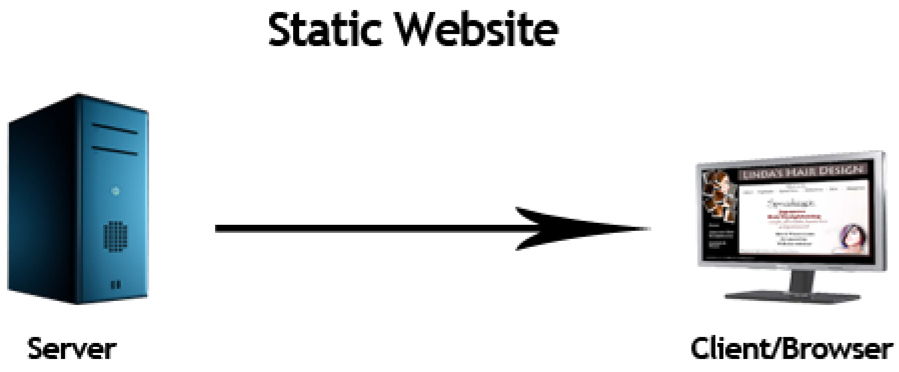
A website can be of two types:

- Static Website
- Dynamic Website

Static website

Static website is the basic type of website that is easy to create. You don't need the knowledge of web programming and database design to create a static website. Its web pages are coded in HTML.

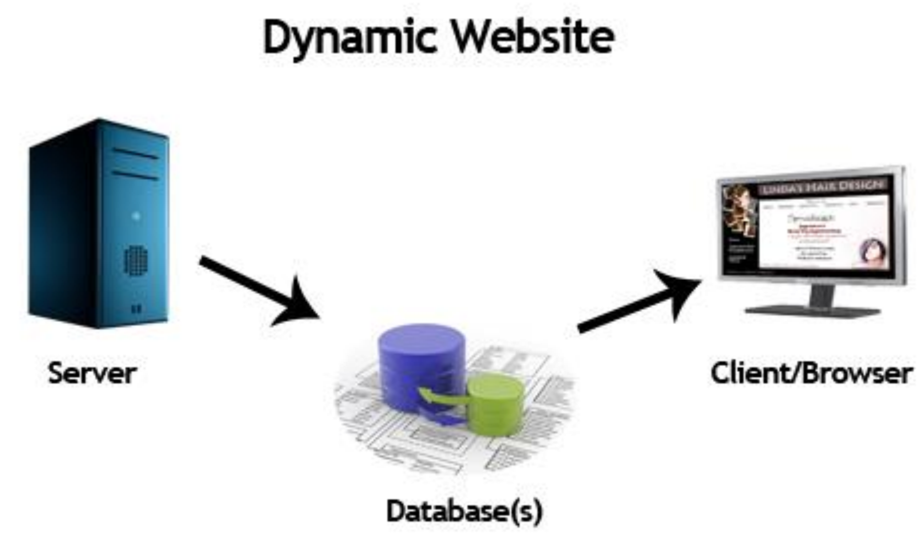
The codes are fixed for each page so the information contained in the page does not change and it looks like a printed page.



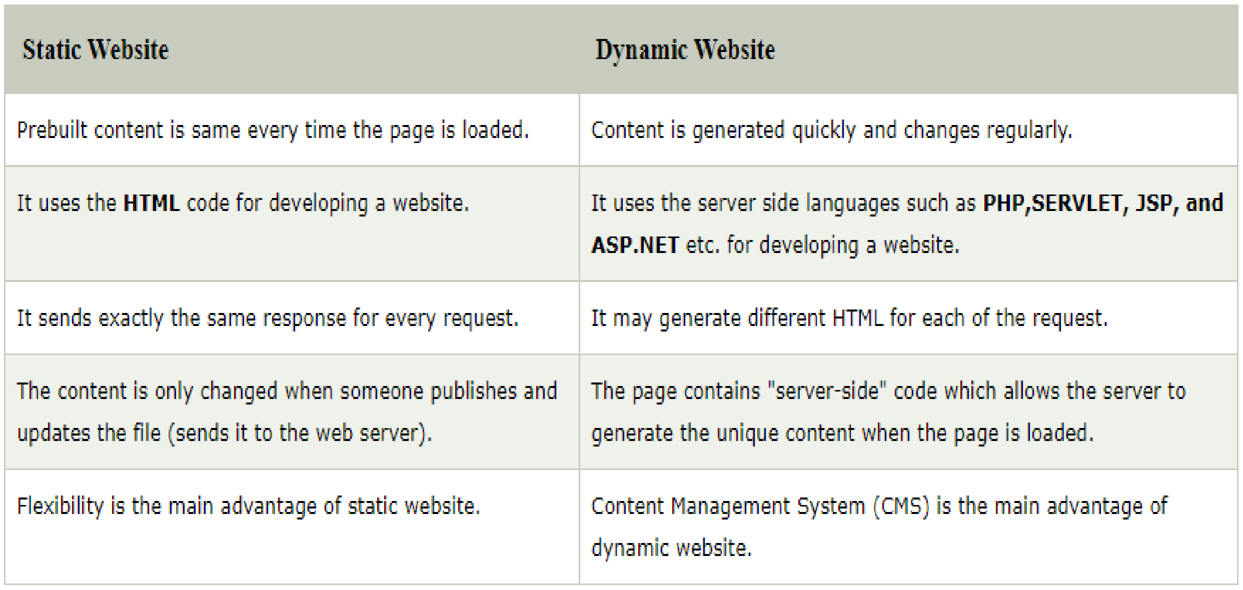
Dynamic website

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

Dynamic website uses client-side scripting or server-side scripting, or both to generate dynamic content. Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.



Static vs Dynamic website

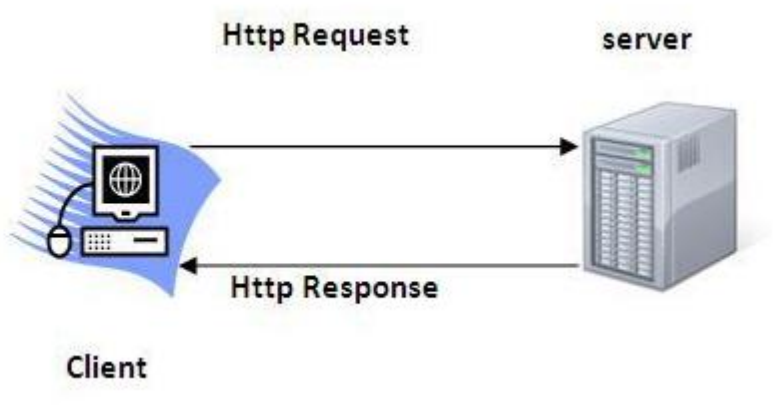


Static Website	Dynamic Website
Prebuilt content is same every time the page is loaded.	Content is generated quickly and changes regularly.
It uses the **HTML** code for developing a website.	It uses the server side languages such as **PHP,SERVLET, JSP, and ASP.NET** etc. for developing a website.
It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

HTTP (Hyper Text Transfer Protocol)

The Hypertext Transfer Protocol (HTTP) is application-level protocol for collaborative, distributed, hypermedia information systems. It is the data communication protocol used to establish communication between client and server.

HTTP is TCP/IP based communication protocol, which is used to deliver the data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port is TCP 80. It provides the standardized way for computers to communicate with each other.



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

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

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

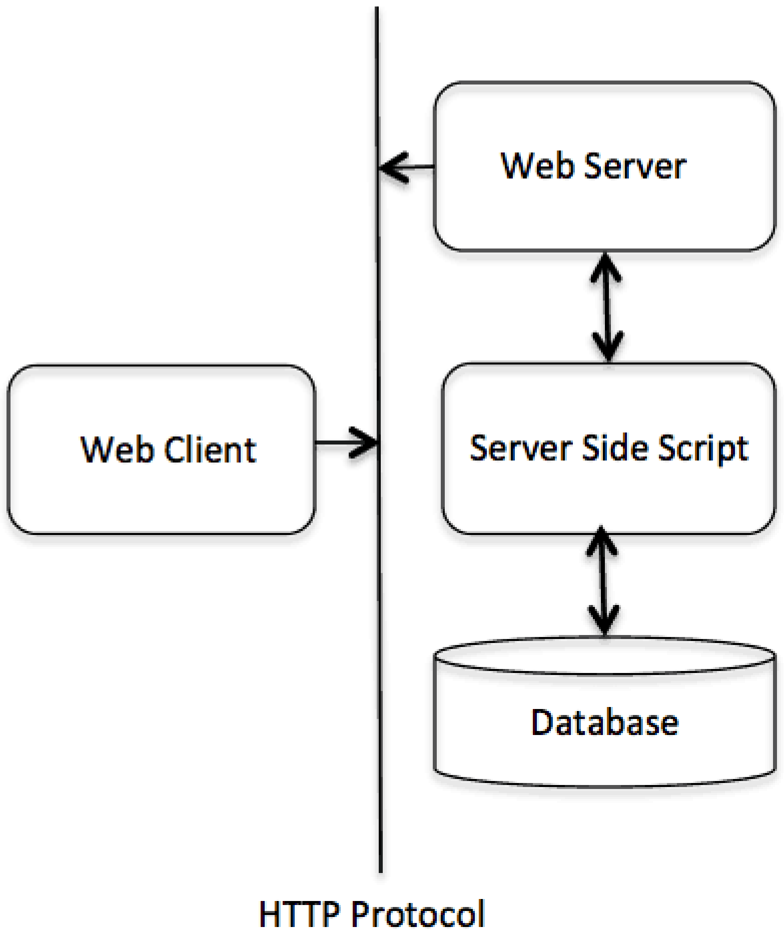
There are three fundamental features that make the HTTP a simple and powerful protocol used for communication:

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

- **HTTP is connectionless:** It is a connectionless approach in which HTTP client i.e., a browser initiates the HTTP request and after the request is sent the client disconnects from server and waits for the response.
- **HTTP is stateless:** The client and server are aware of each other during a current request only. Afterwards, both of them forget each other. Due to the stateless nature of protocol, neither the client nor the server can retain the information about different request across the web pages.

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

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



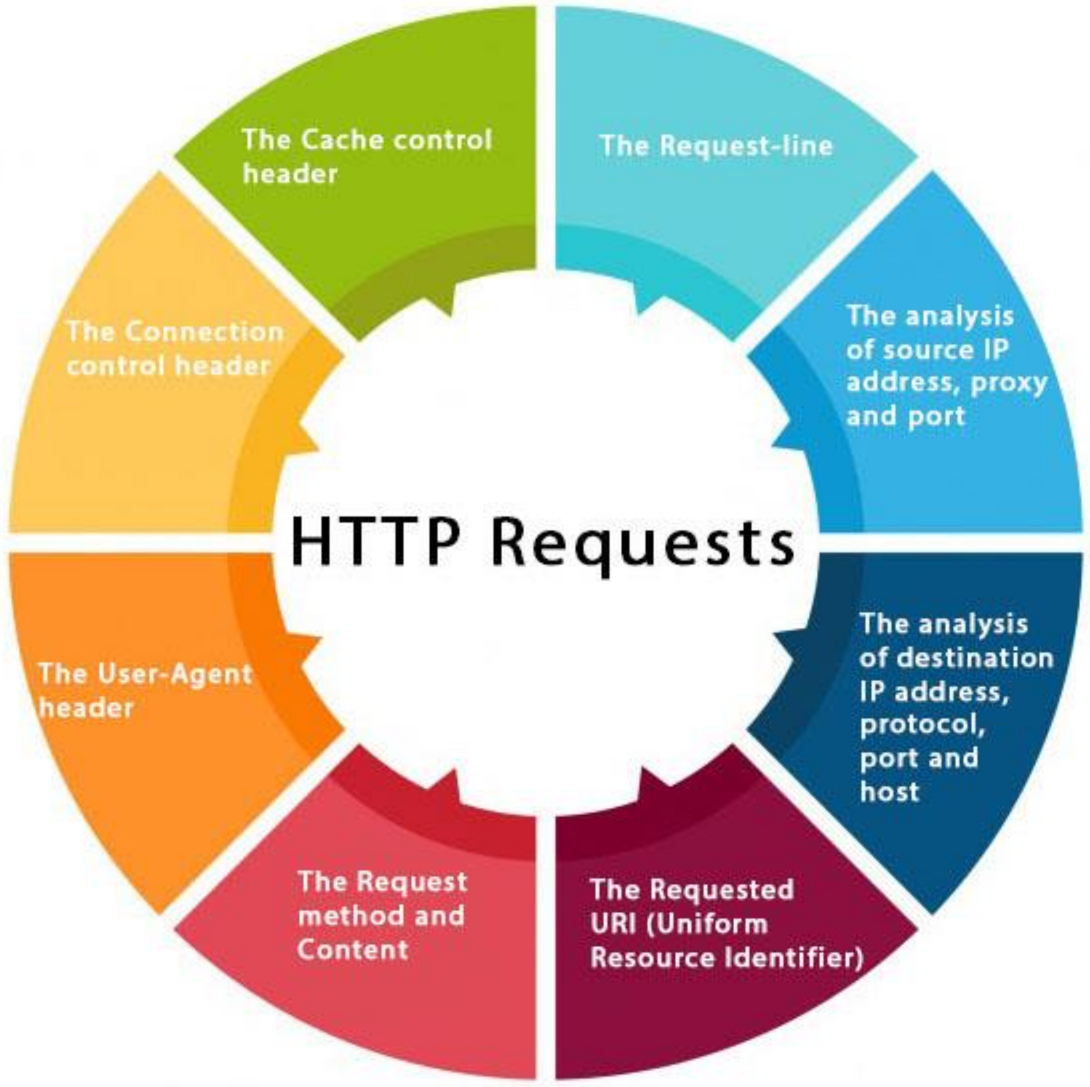
HTTP is request/response protocol which is based on client/server based architecture. In this protocol, web browser, search engines, etc. behave as HTTP clients and the Web server like Servlet behaves as a server

HTTP Requests

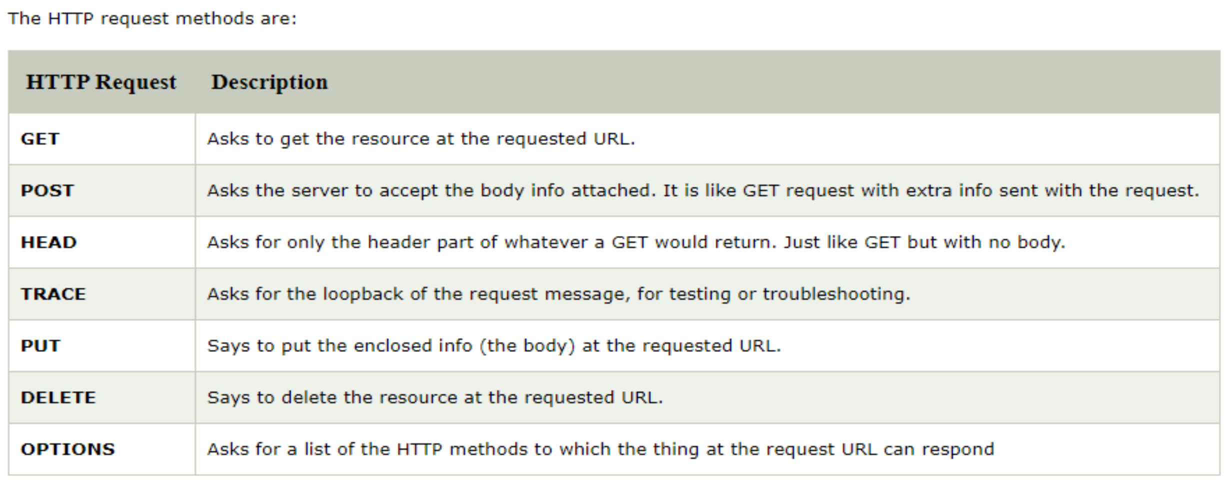
The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

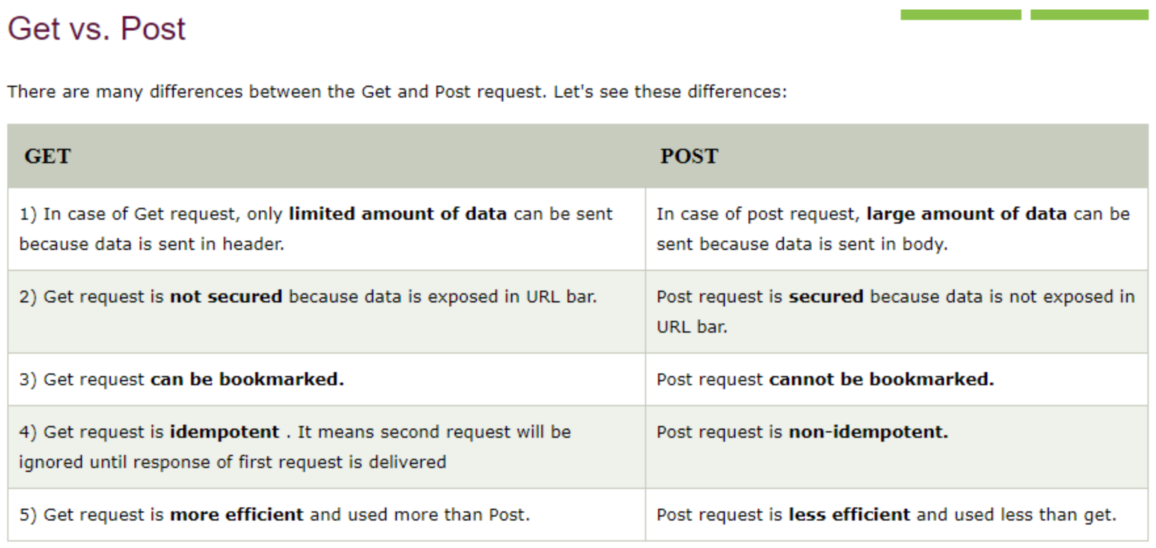
- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header


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

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


The HTTP request methods are:

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


Get vs. Post

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

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

GET and POST

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

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

Anatomy of Get Request

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

Some other features of GET requests are:

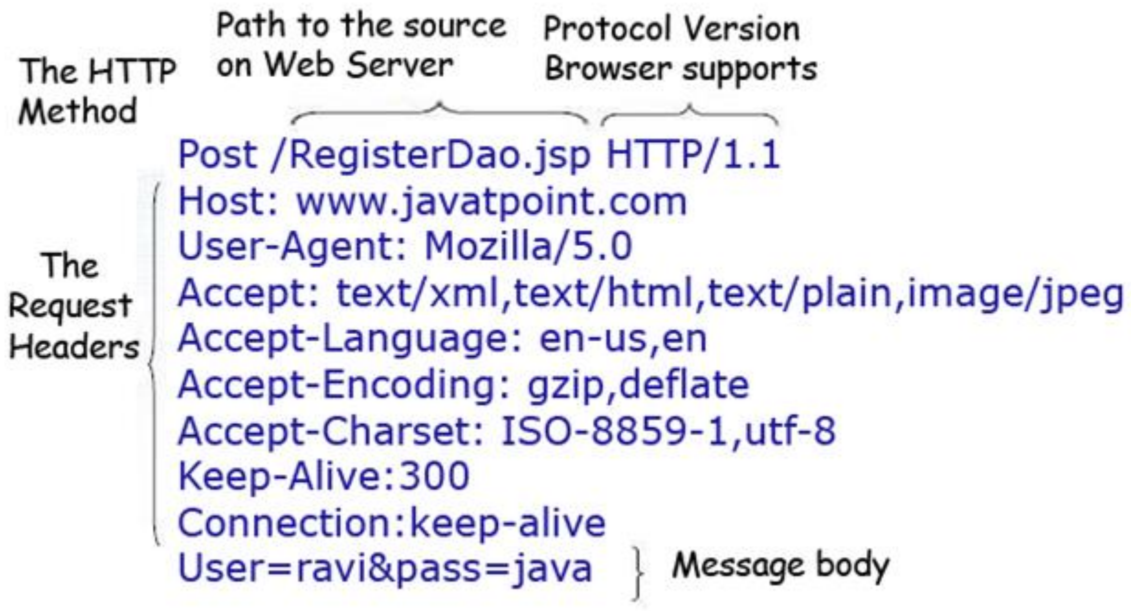
- It remains in the browser history
- It can be bookmarked
- It can be cached
- It have length restrictions
- It should never be used when dealing with sensitive data
- It should only be used for retrieving the data

Anatomy of Post Request

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

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

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


The HTTP Method
Path to the source on Web Server
Protocol Version Browser supports
The Request Headers
Message body

```
Post /RegisterDao.jsp HTTP/1.1
Host: www.javatpoint.com
User-Agent: Mozilla/5.0
Accept: text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive:300
Connection:keep-alive
User=ravi&pass=java
```

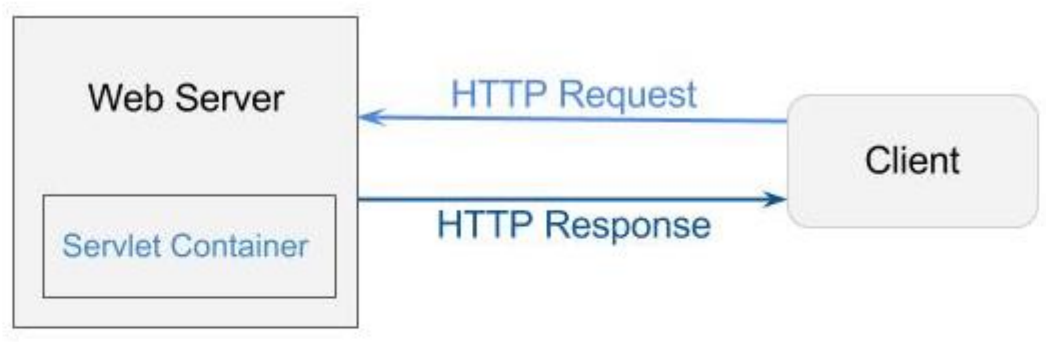
Some other features of POST requests are:

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

Servlet Container

It provides the runtime environment for JavaEE (j2ee) applications. The client/user can request only a static WebPages from the server. If the user wants to read the web pages as per input then the servlet container is used in java.

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



Servlet Container States

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

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

The Servlet Container performs many operations that are given below:

- Life Cycle Management
- Multithreaded support
- Object Pooling
- Security etc.

Server: Web vs. Application

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

There are two types of servers:

1. Web Server
2. Application Server

Web Server

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

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

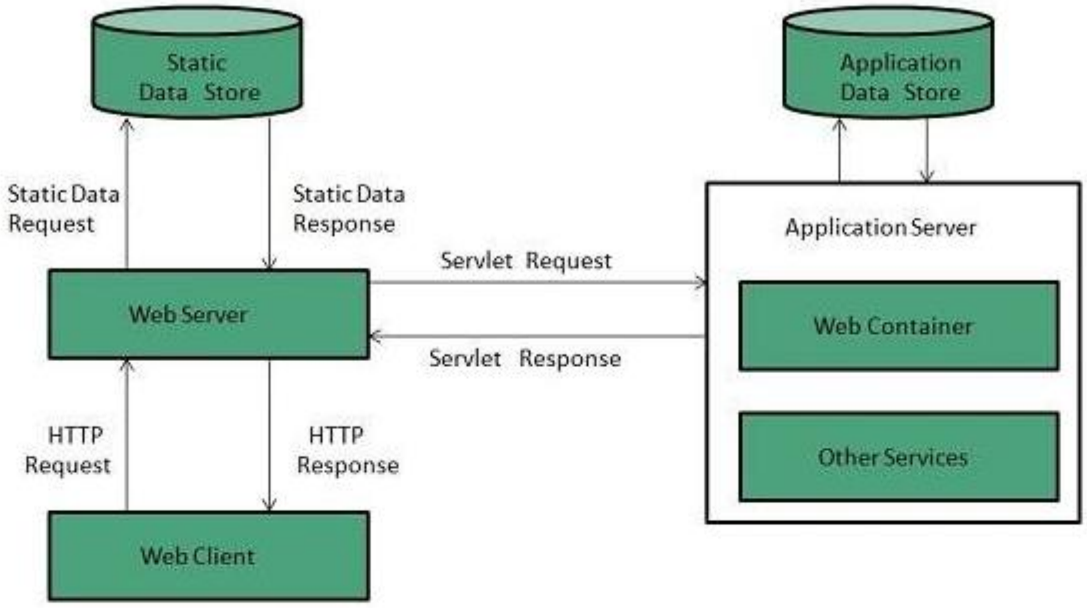
Examples of Web Servers are: **Apache Tomcat** and **Resin**.

Web Server Working

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

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

The block diagram representation of Web Server is shown below:

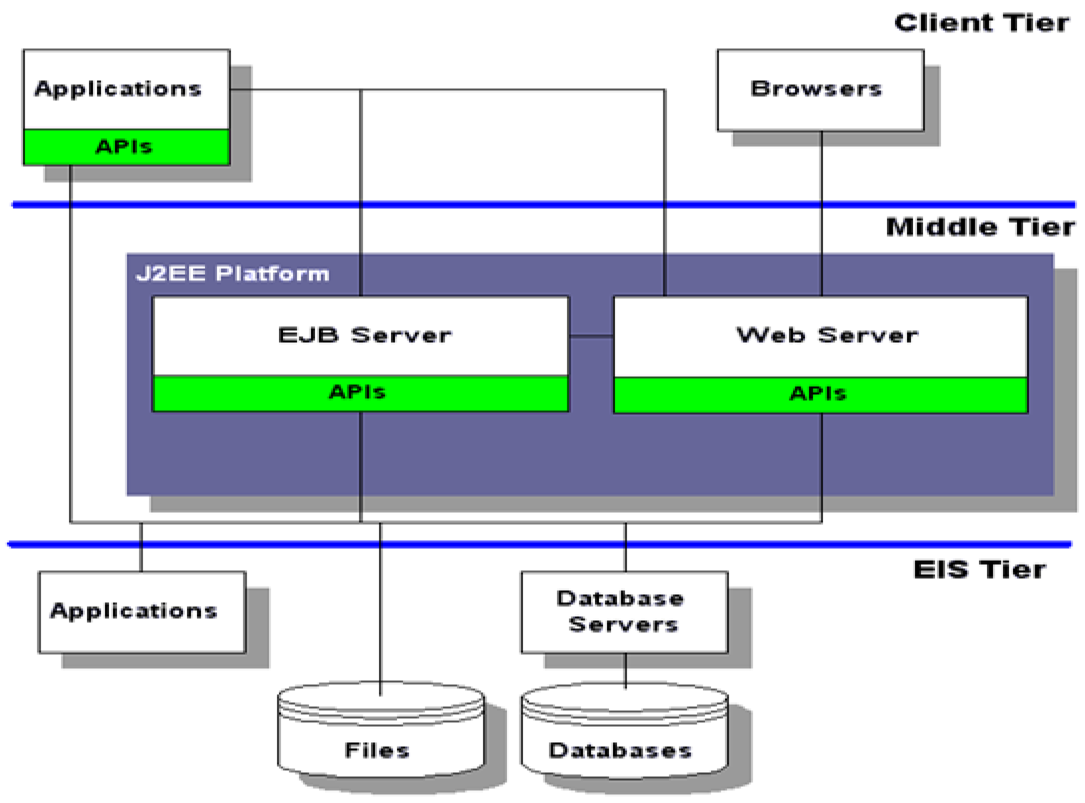


Important points

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

Application Server

- Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture.
- It provides the middleware services for state maintenance and security, along with persistence and data access. It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.
- The block diagram representation of Application Server is shown below:



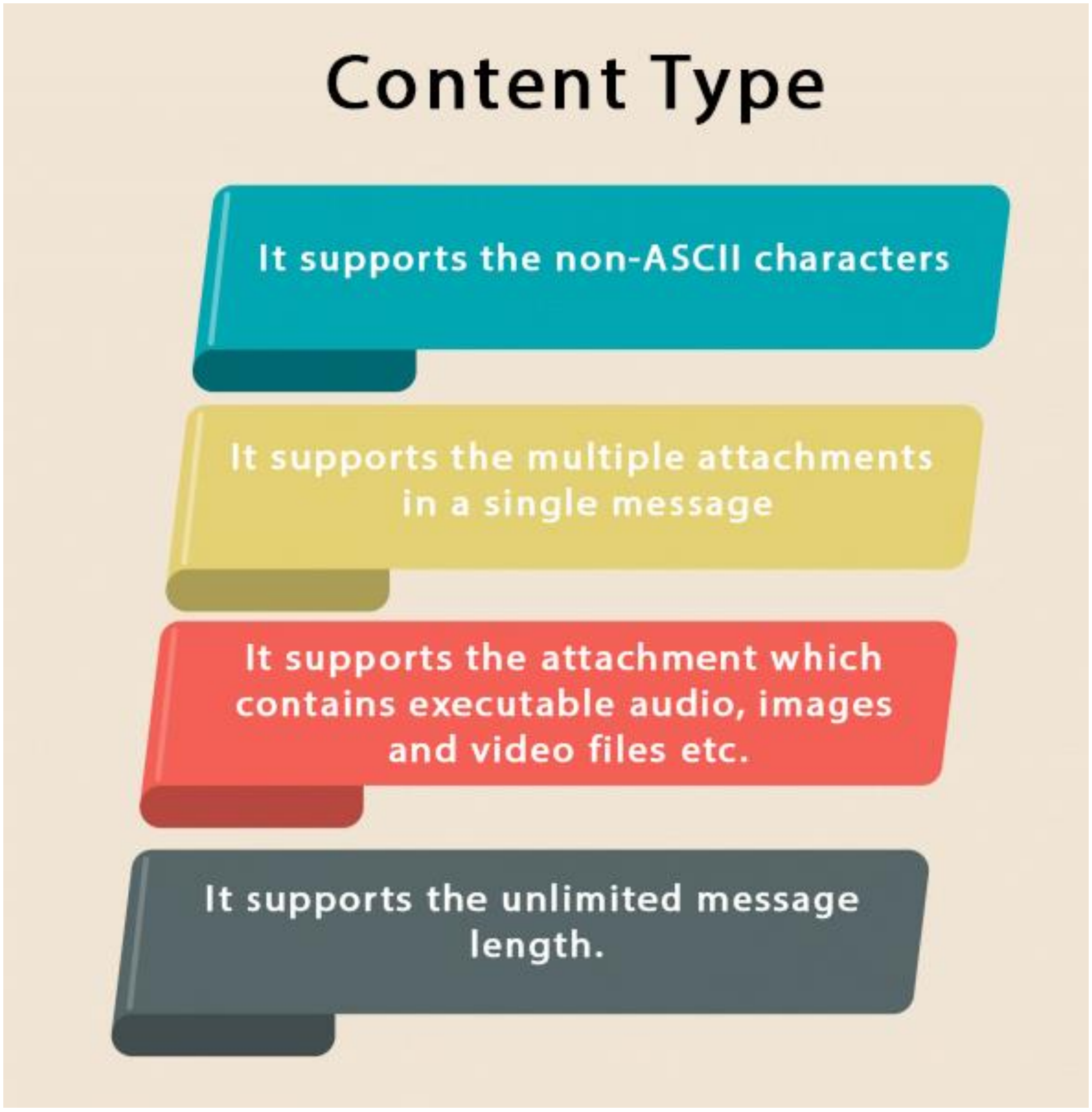
The Example of Application Servers are:

1. **JBoss:** Open-source server from JBoss community.
2. **Glassfish:** Provided by Sun Microsystem. Now acquired by Oracle.
3. **Weblogic:** Provided by Oracle. It more secured.
4. **Websphere:** Provided by IBM.

Content Type

- Content Type is also known as **MIME (Multipurpose internet Mail Extension)**Type. It is a **HTTP header** that provides the description about what are you sending to the browser.
- MIME is an internet standard that is used for extending the limited capabilities of email by allowing the insertion of sounds, images and text in a message.
- The features provided by MIME to the email services are as given below:
- It supports the non-ASCII characters

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


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

List of Content Types

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

- text/html
- text/plain
- application/msword
- application/vnd.ms-excel
- application/jar
- application/pdf
- application/octet-stream
- application/x-zip
- images/jpeg
- images/png
- images/gif
- audio/mp3
- video/mp4
- video/quicktime etc.

Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

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

Interfaces in javax.servlet package

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

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher

5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

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

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

Interfaces in javax.servlet.http package

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

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession

4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

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

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

Servlet Interface

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

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

Methods of Servlet interface

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

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

Servlet Example by implementing Servlet interface

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

File: First.java

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

public class First implements Servlet{
ServletConfig config=null;

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

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

res.setContentType("text/html");

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

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

}
```

GenericServlet class

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

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

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

Methods of GenericServlet class

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

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

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

Servlet Example by inheriting the GenericServlet class

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

File: First.java

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

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

res.setContentType("text/html");

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

}
}
```

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:

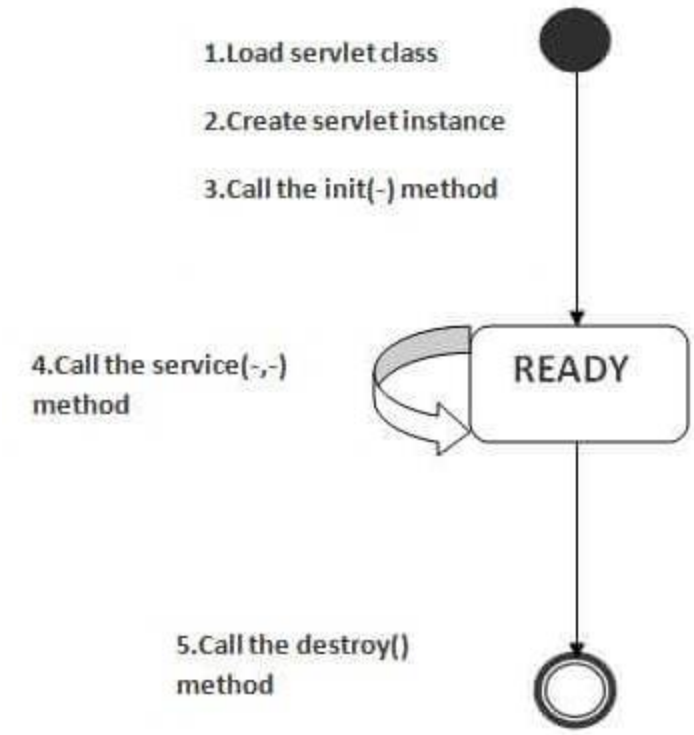
1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.

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.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

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:

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

4) service method is invoked

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

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

5) destroy method is invoked

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

```
public void destroy()
```

Steps to create a servlet example

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

The servlet example can be created by three ways:

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

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

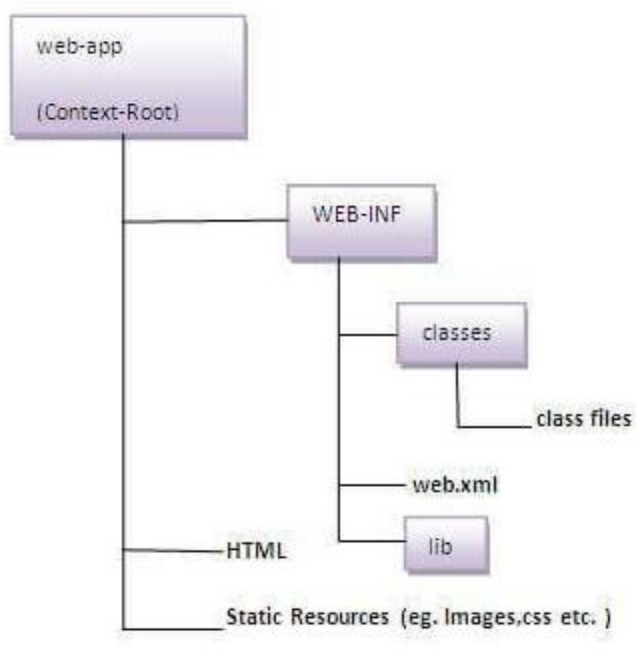
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

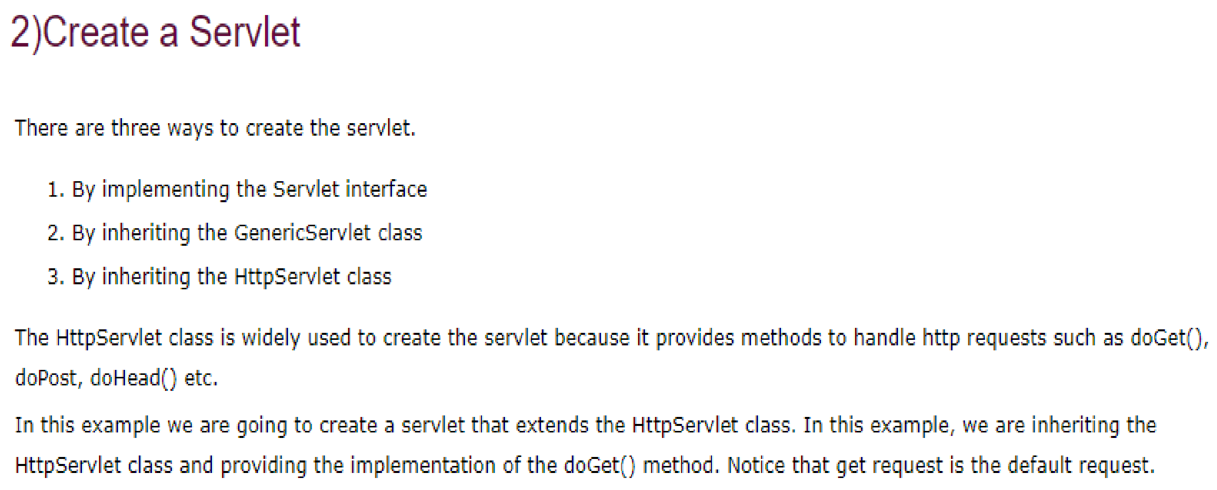
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1)Create a directory structures

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

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





2)Create a Servlet

There are three ways to create the servlet.

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

The HttpServlet class is widely used to create the servlet because it provides methods to handle http 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

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

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

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

3)Compile the servlet

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

Jar file	Server

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

Two ways to load the jar file

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

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

---

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

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

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

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

web.xml file

```
<web-app>

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

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

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

</web-app>
```

Description of the elements of web.xml file

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

<web-app> represents the whole application.

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

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

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

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

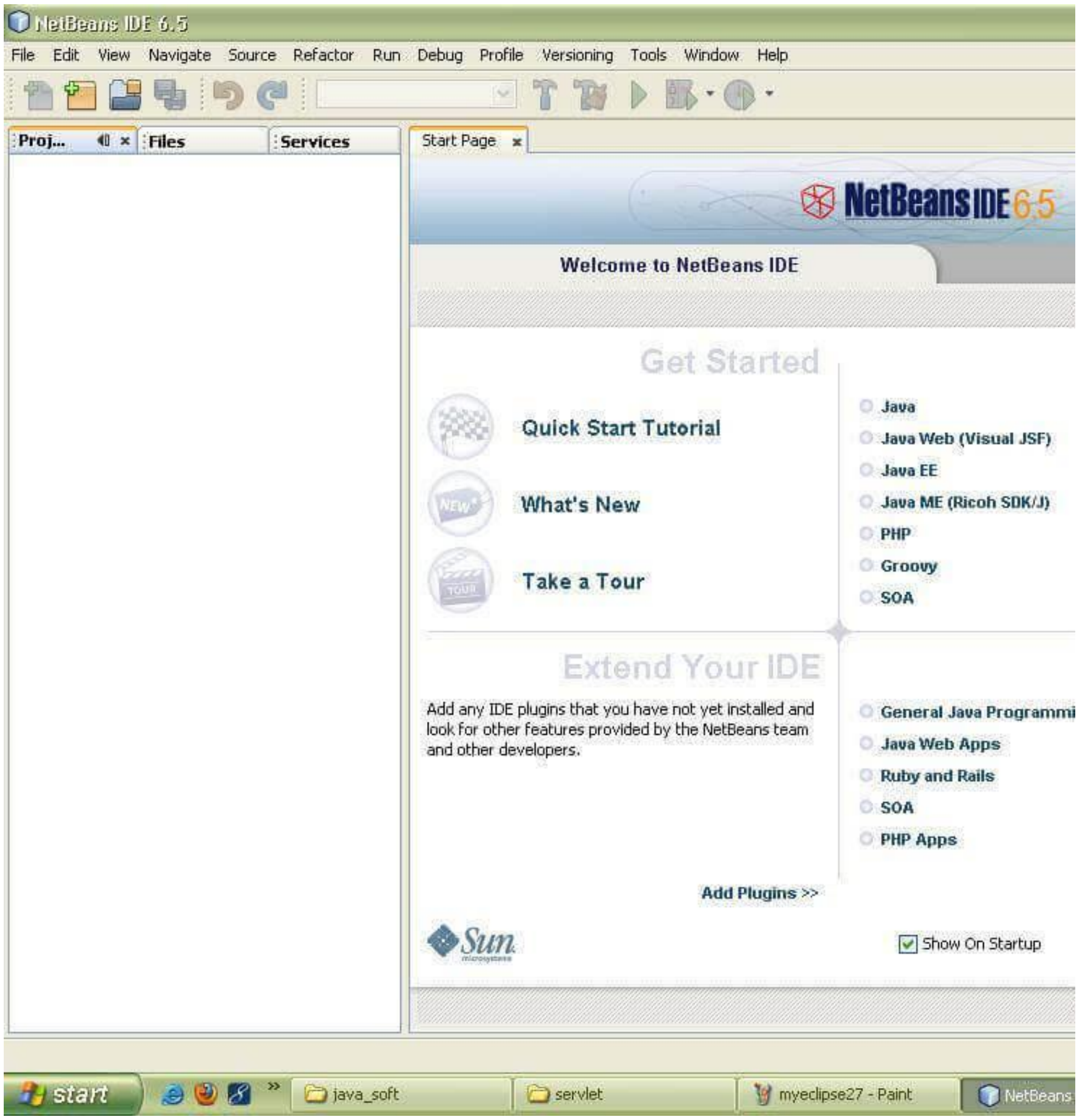
<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke th

---

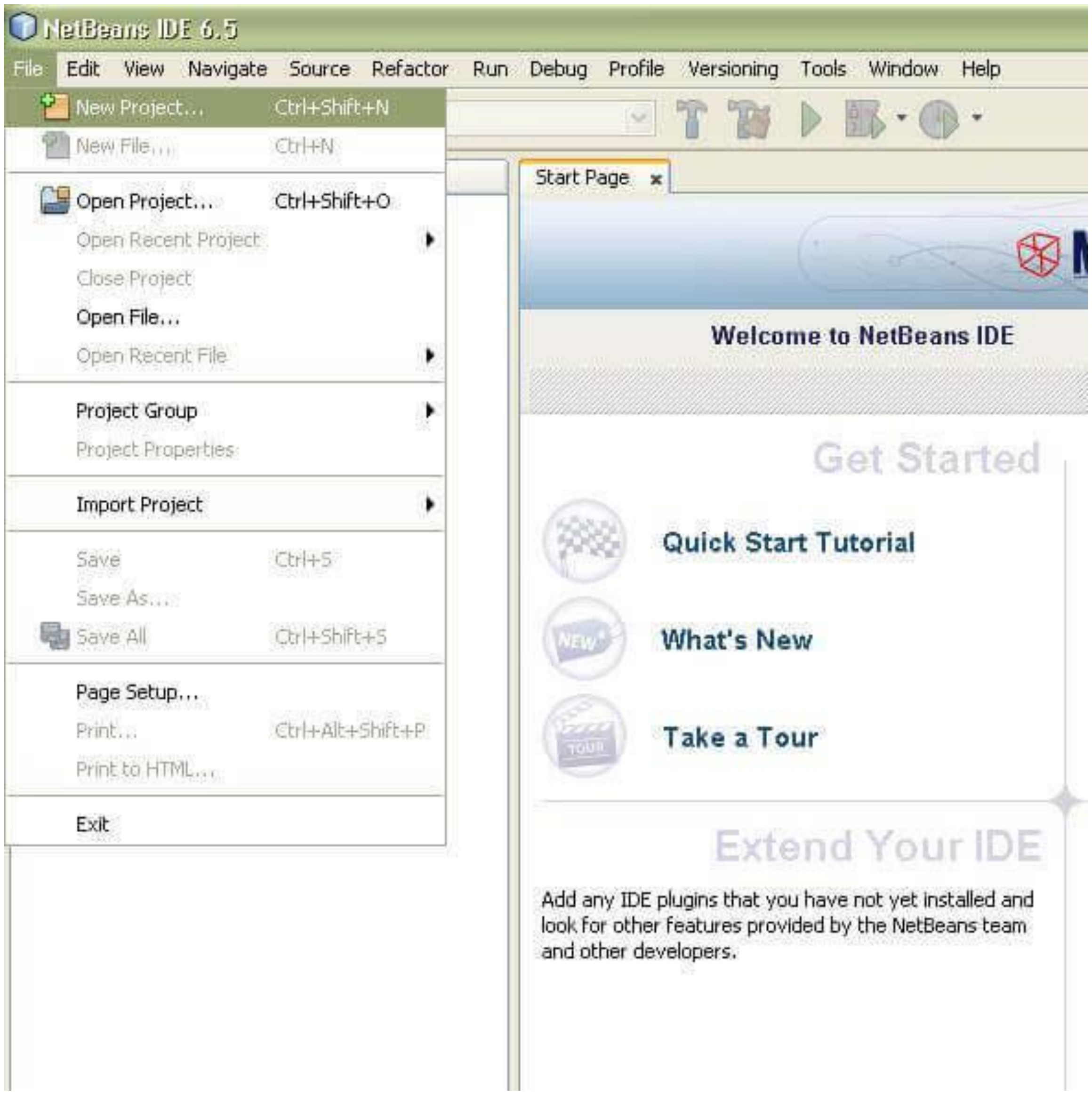
5)Start the Server and deploy the project

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

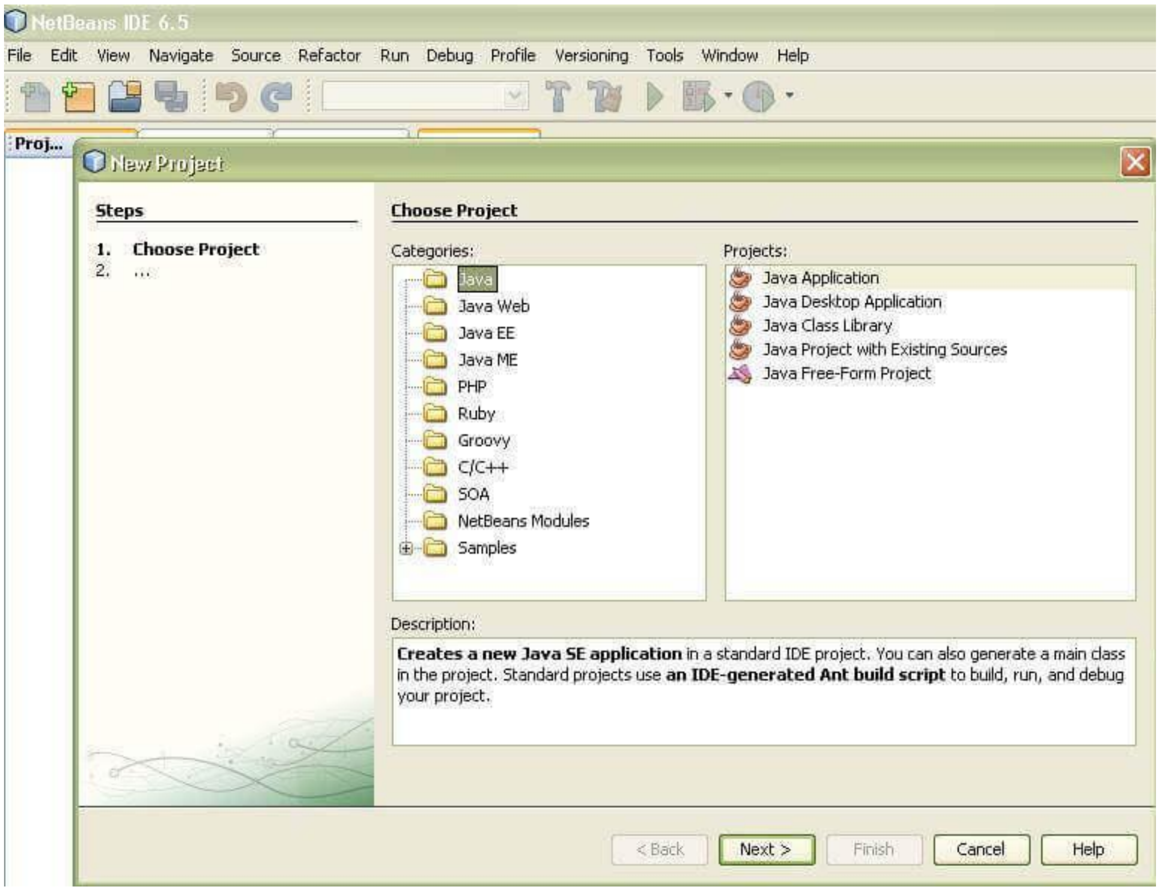
Creating a servlet in NetBeans IDE:



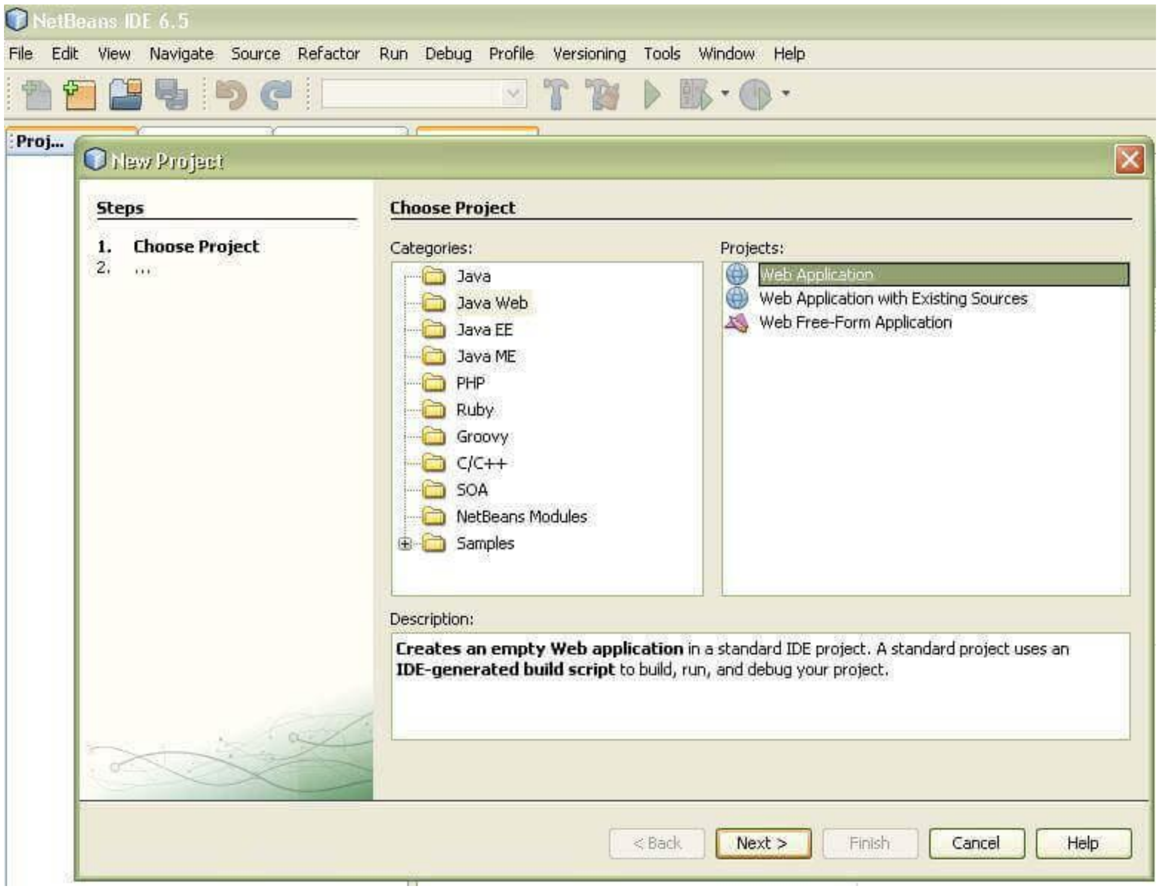
Add any IDE plugins that you have not yet installed and look for other features provided by the NetBeans team and other developers.



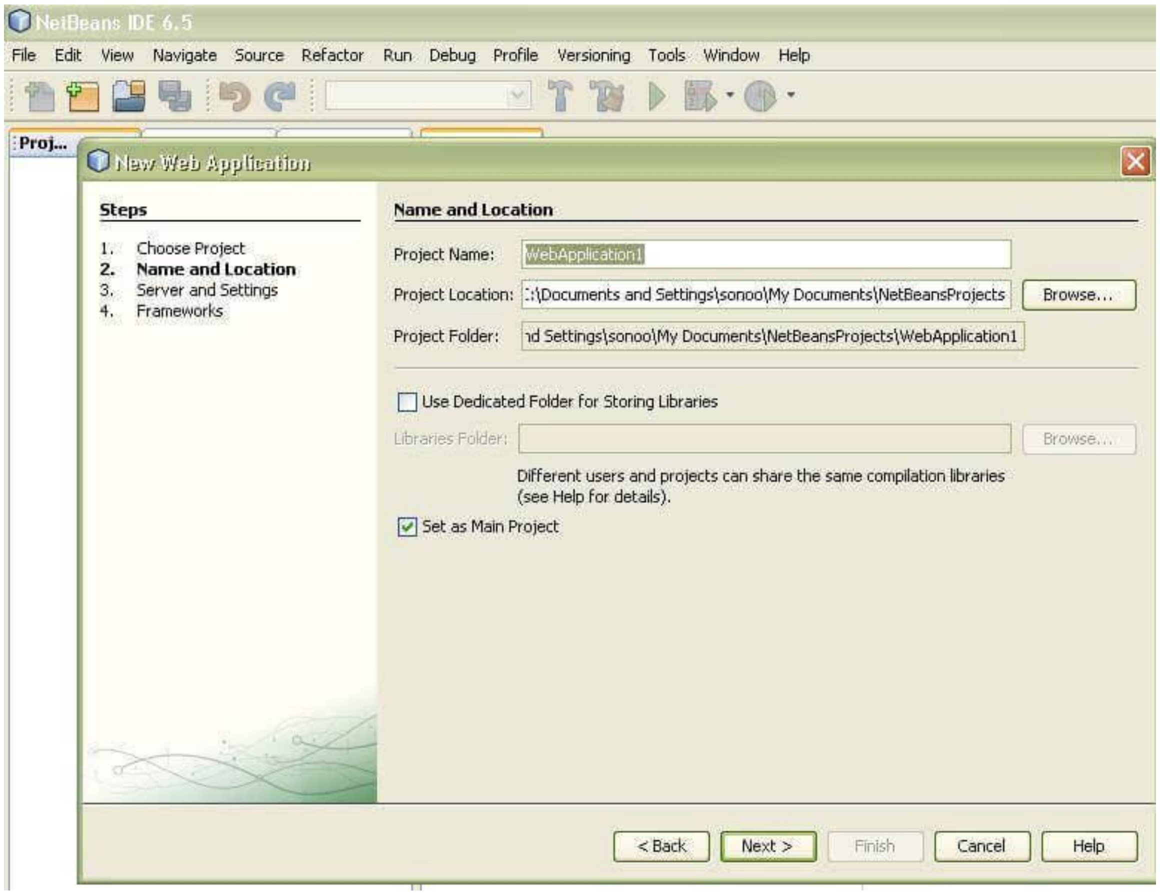
Add any IDE plugins that you have not yet installed and look for other features provided by the NetBeans team and other developers.



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.



Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.


NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Proj...
New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: WebApplication1

Project Location: :\Documents and Settings\sonoo\My Documents\NetBeansProjects

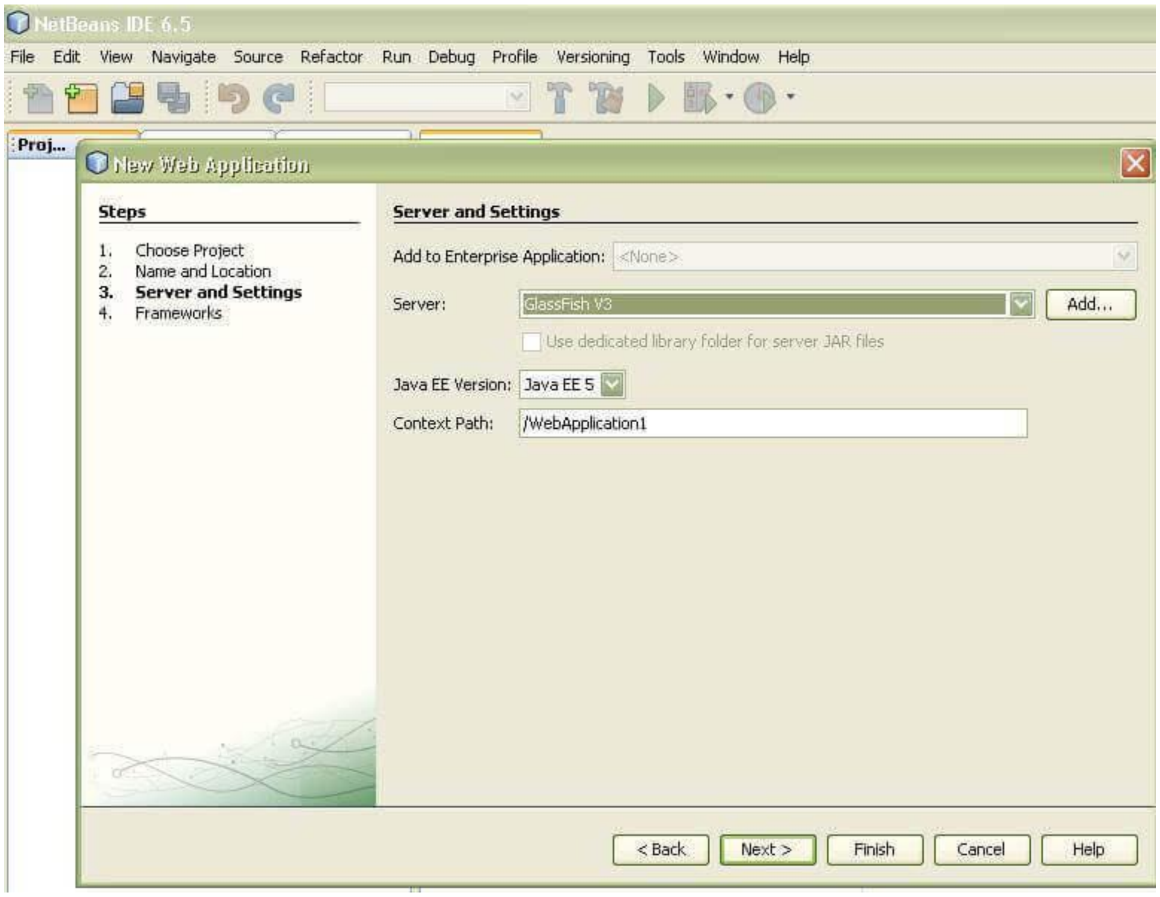
Project Folder: nd Settings\sonoo\My Documents\NetBeansProjects\WebApplication1

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Set as Main Project



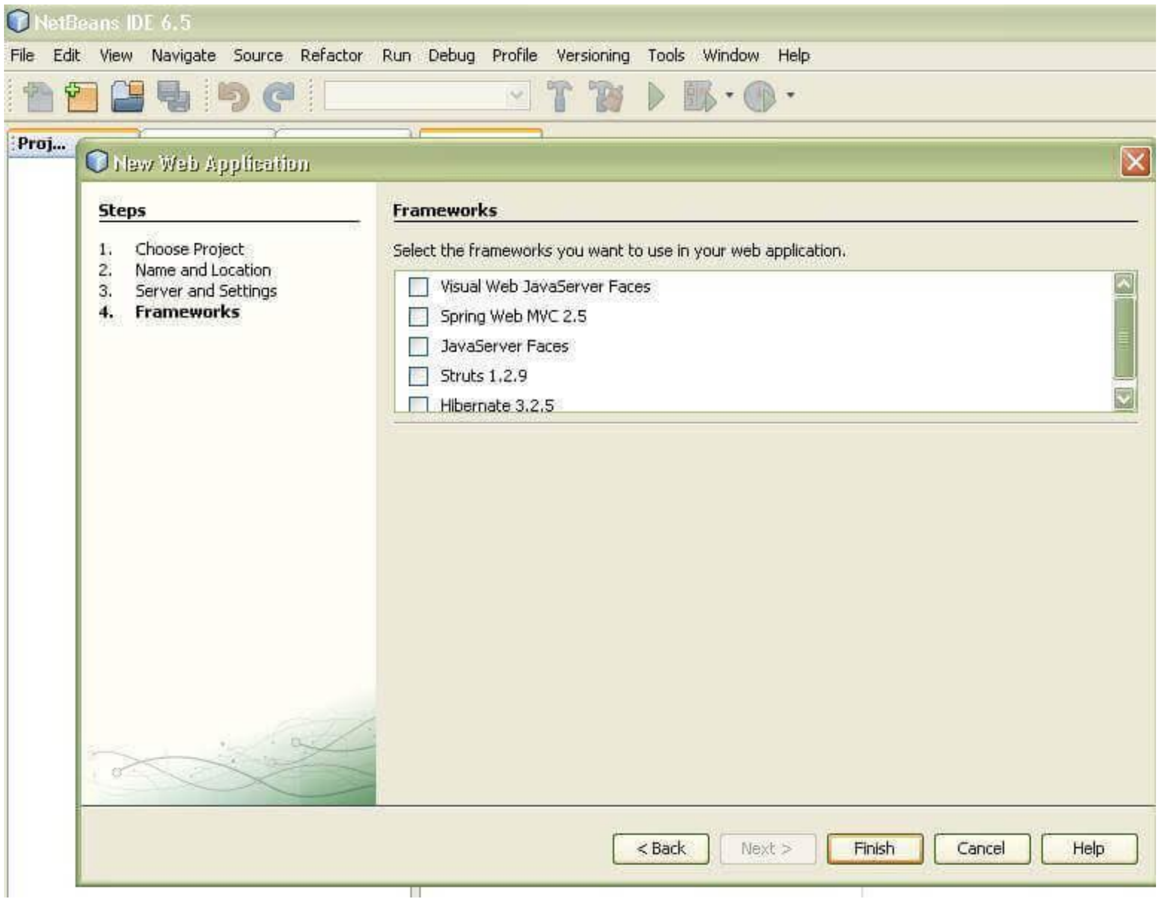
Add to Enterprise Application: <None>

Server: GlassFish V3

Use dedicated library folder for server JAR files

Java EE Version: Java EE 5

Context Path: /WebApplication1


NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Proj...
New Web Application

Steps

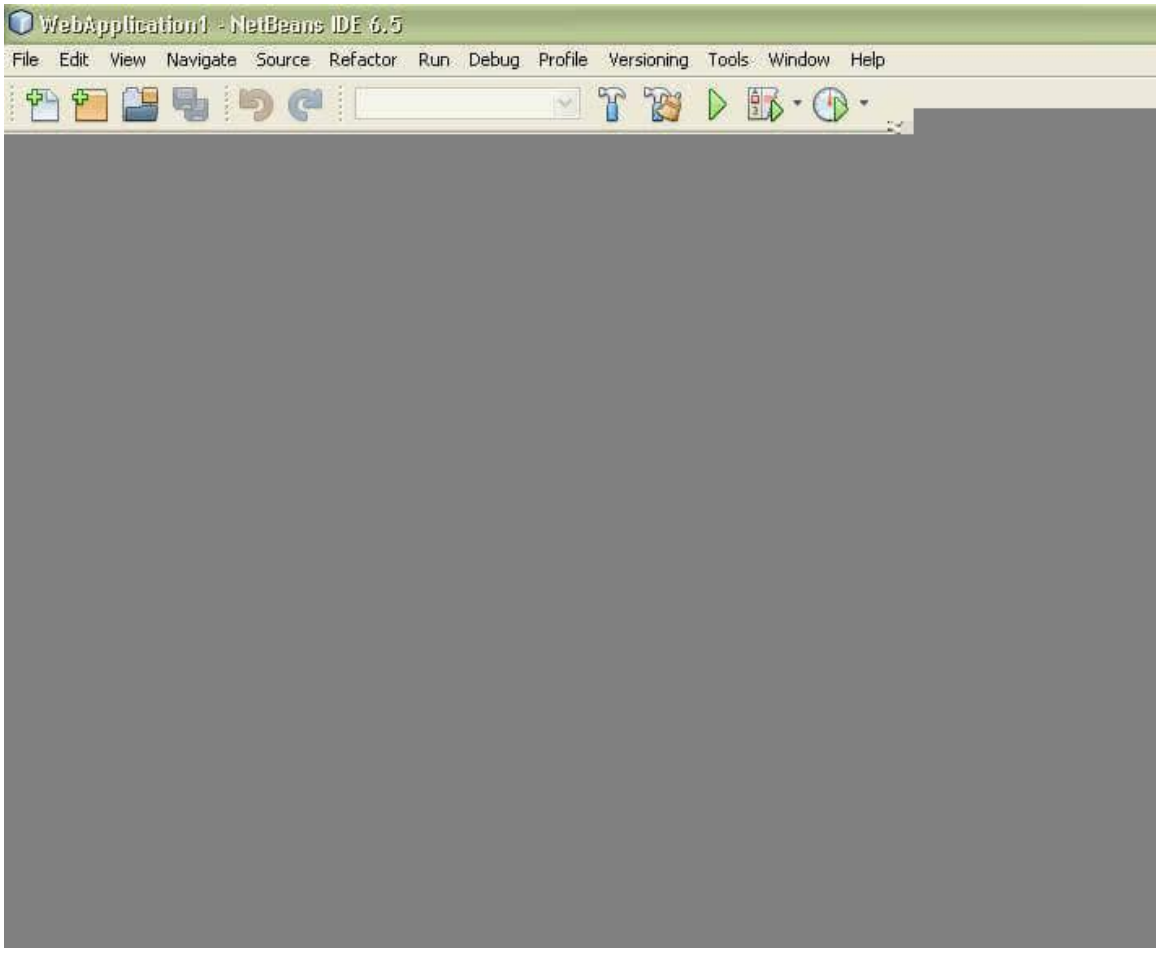
1. Choose Project
2. Name and Location
3. Server and Settings
4. **Frameworks**

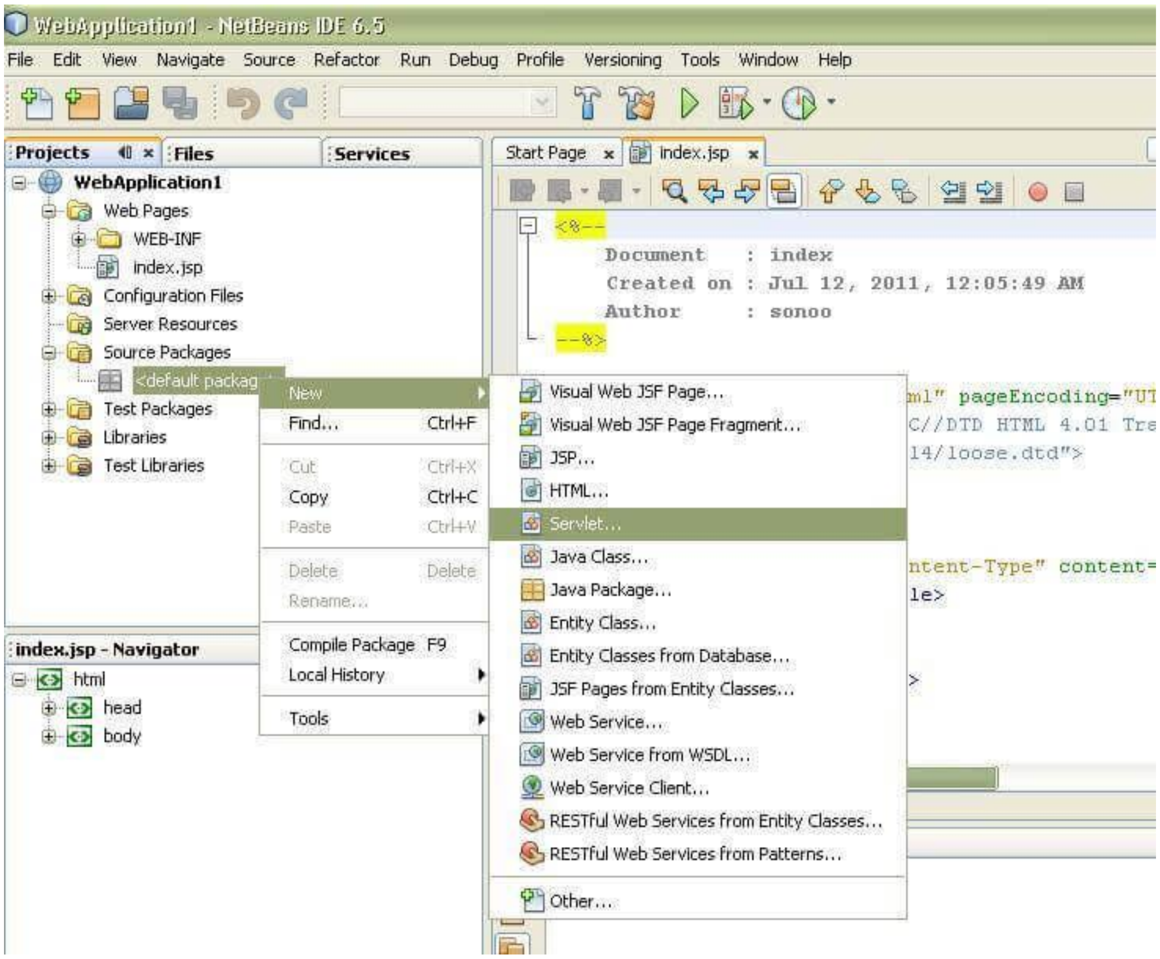
Frameworks

Select the frameworks you want to use in your web application.

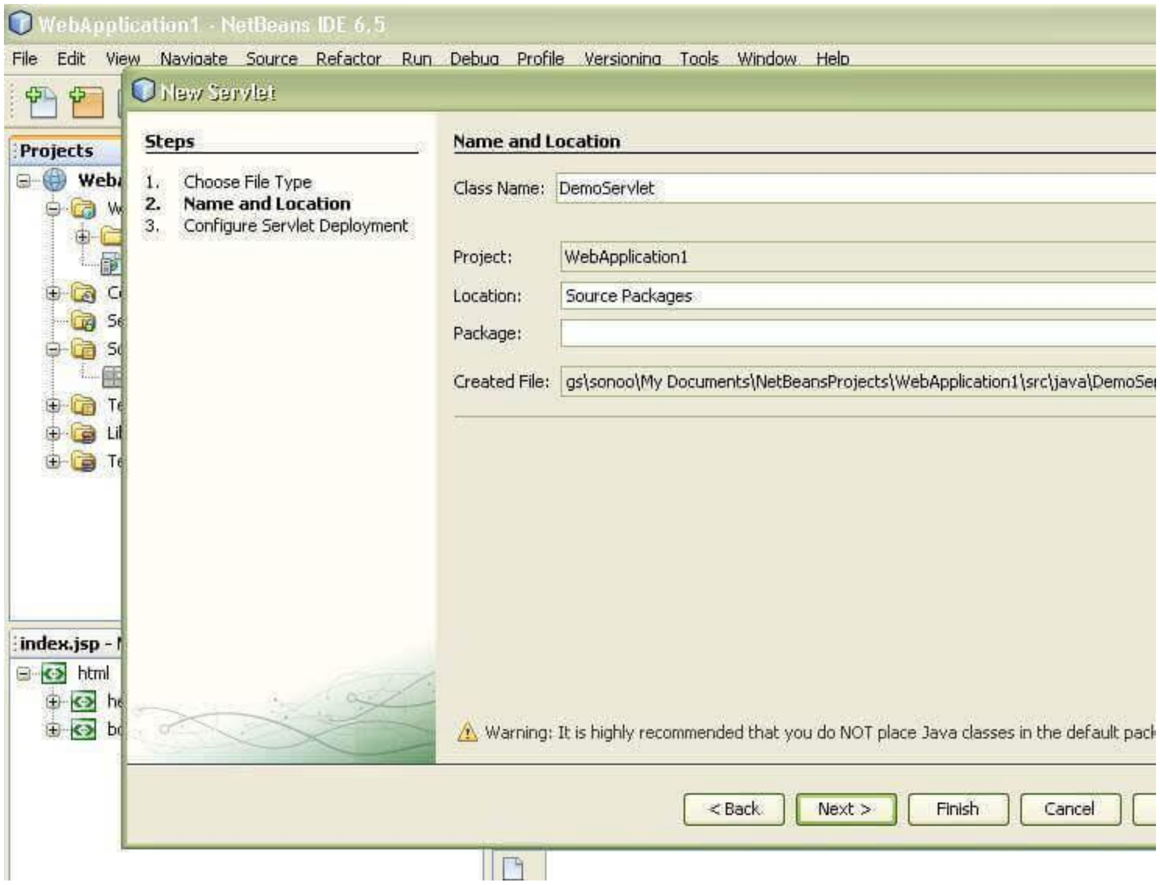
- [ ] Visual Web JavaServer Faces
- [ ] Spring Web MVC 2.5
- [ ] JavaServer Faces
- [ ] Struts 1.2.9
- [ ] Hibernate 3.2.5

< Back
Next >
Finish
Cancel
Help


WebApplication1 - NetBeans IDE 6.5
File
Edit
View
Navigate
Source
Refactor
Run
Debug
Profile
Versioning
Tools
Window
Help



```
<%--
    Document   : index
    Created on : Jul 12, 2011, 12:05:49 AM
    Author     : sonoo
--%>
```



WebApplication1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
New Servlet
Projects
index.jsp -
html
< Back
Next >
Finish
Cancel

Steps

1. Choose File Type
2. **Name and Location**
3. Configure Servlet Deployment

Name and Location

Class Name: DemoServlet

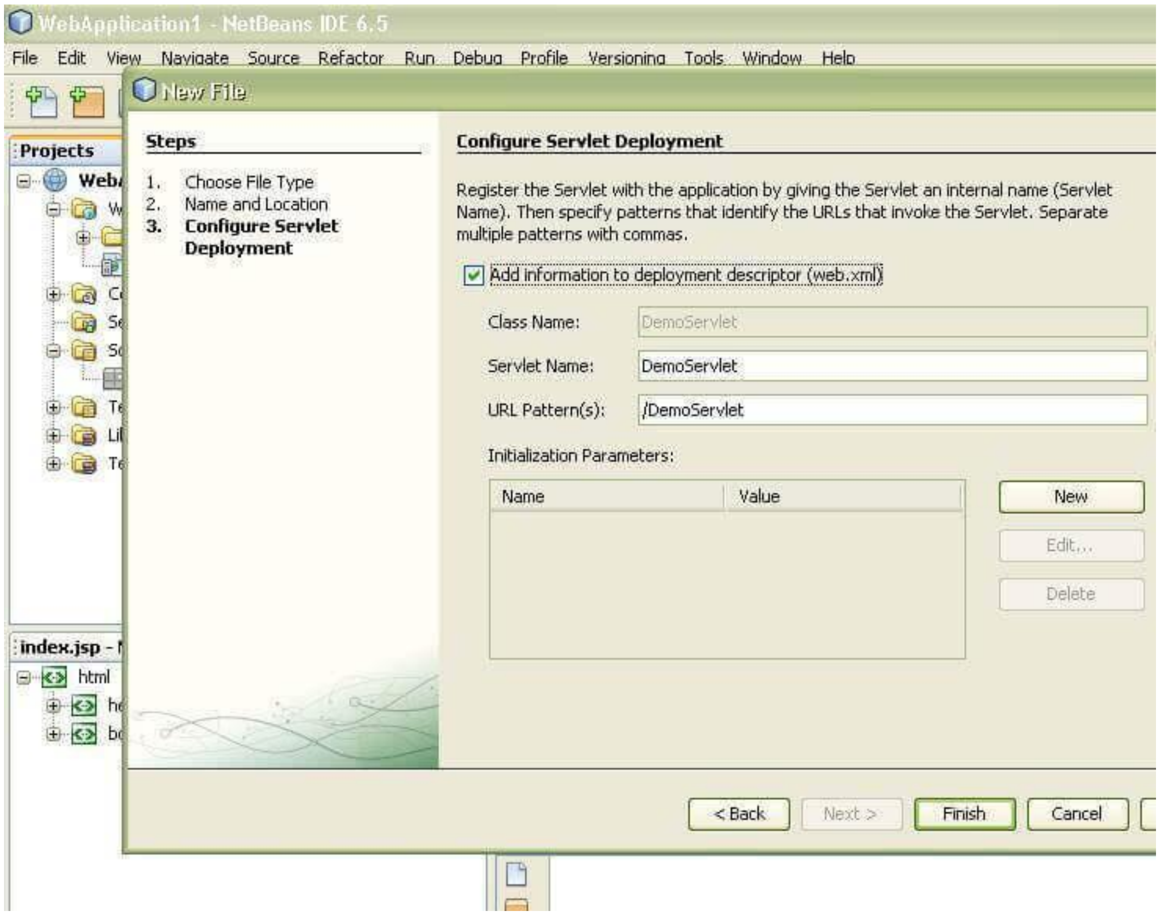
Project: WebApplication1

Location: Source Packages

Package:

Created File: gs\sonoo\My Documents\NetBeansProjects\WebApplication1\src\java\DemoSer

Warning: It is highly recommended that you do NOT place Java classes in the default pack



WebApplication1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Projects
index.jsp
html
New File
Steps
1. Choose File Type
2. Name and Location
3. Configure Servlet Deployment
New
Edit...
Delete
< Back
Next >
Finish
Cancel

Configure Servlet Deployment

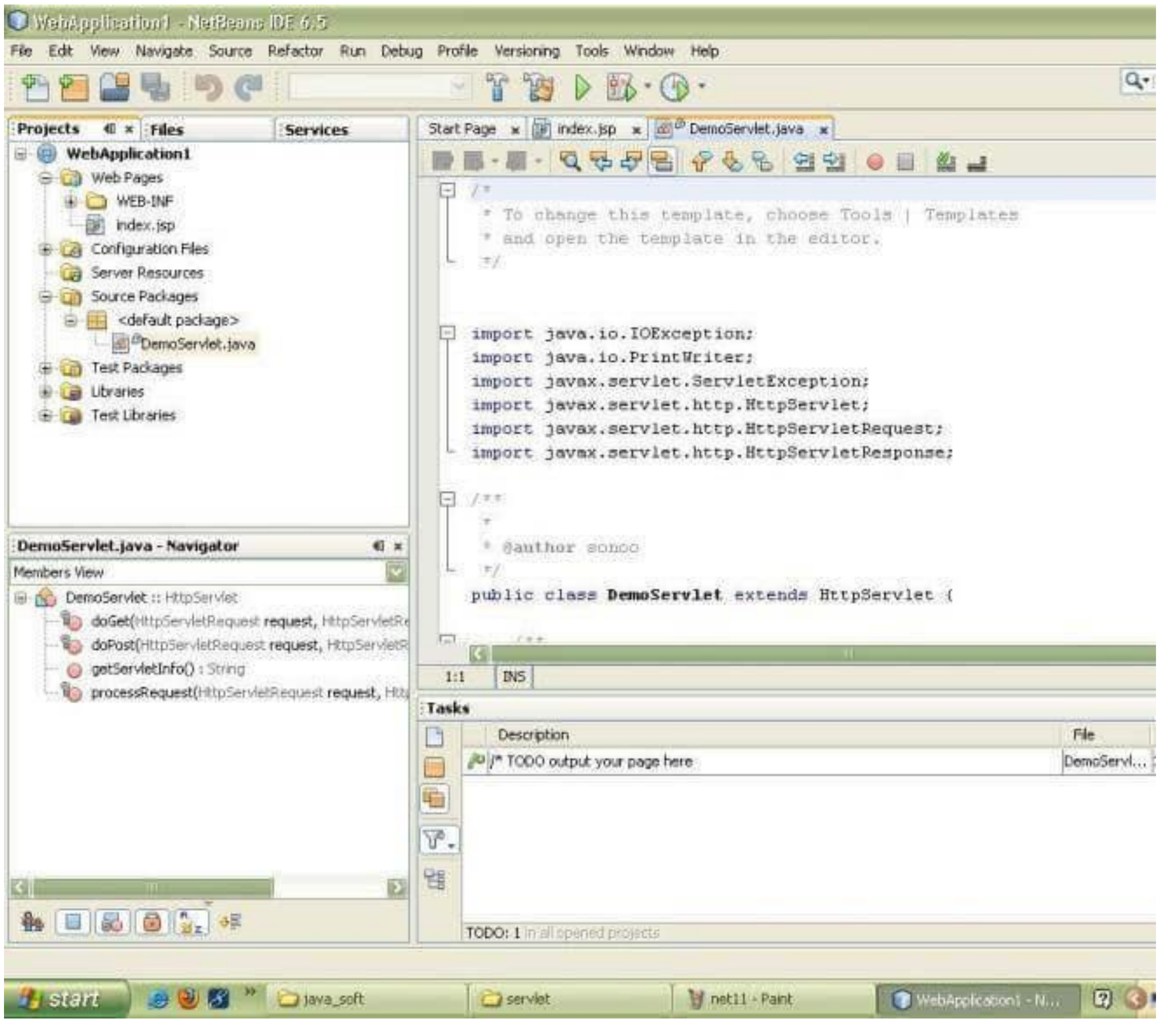
Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

[✓] Add information to deployment descriptor (web.xml)

Class Name:	DemoServlet
Servlet Name:	DemoServlet
URL Pattern(s):	/DemoServlet

Initialization Parameters:

Name	Value

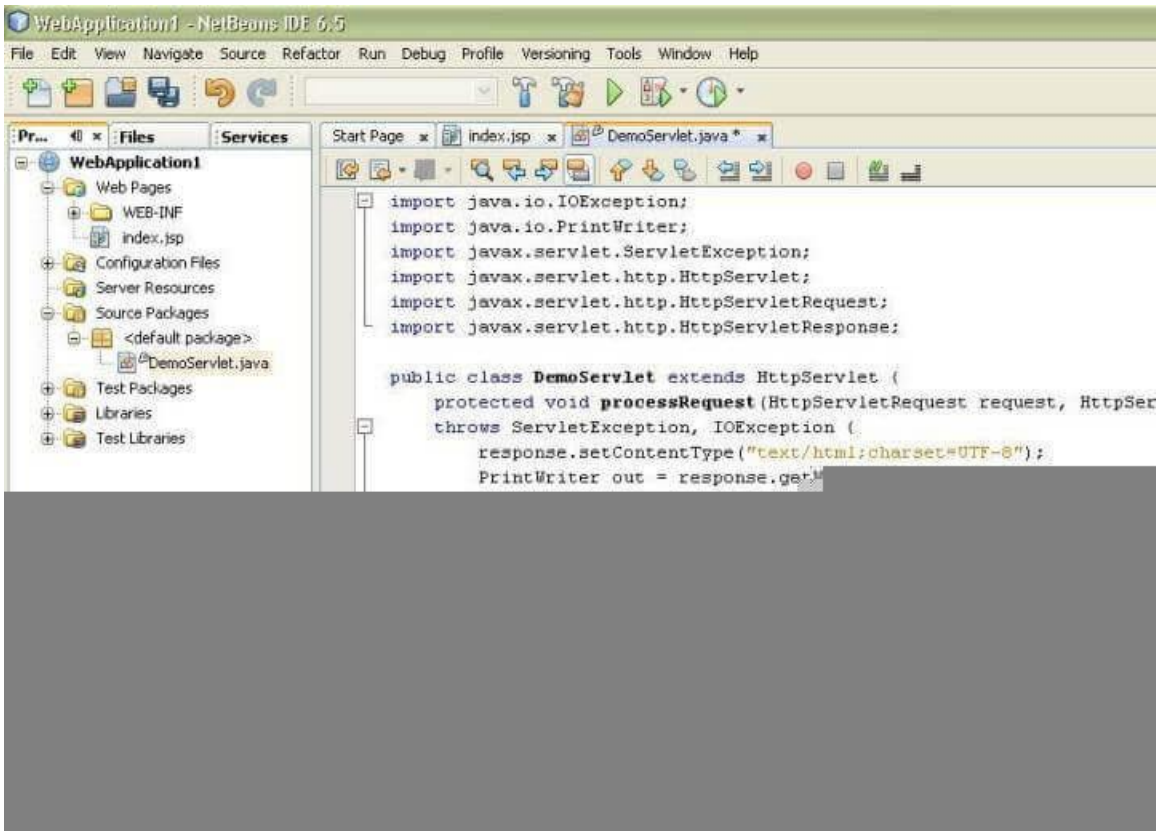


```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */


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;

/**
 *
 * @author sonoo
 */
public class DemoServlet extends HttpServlet {
```

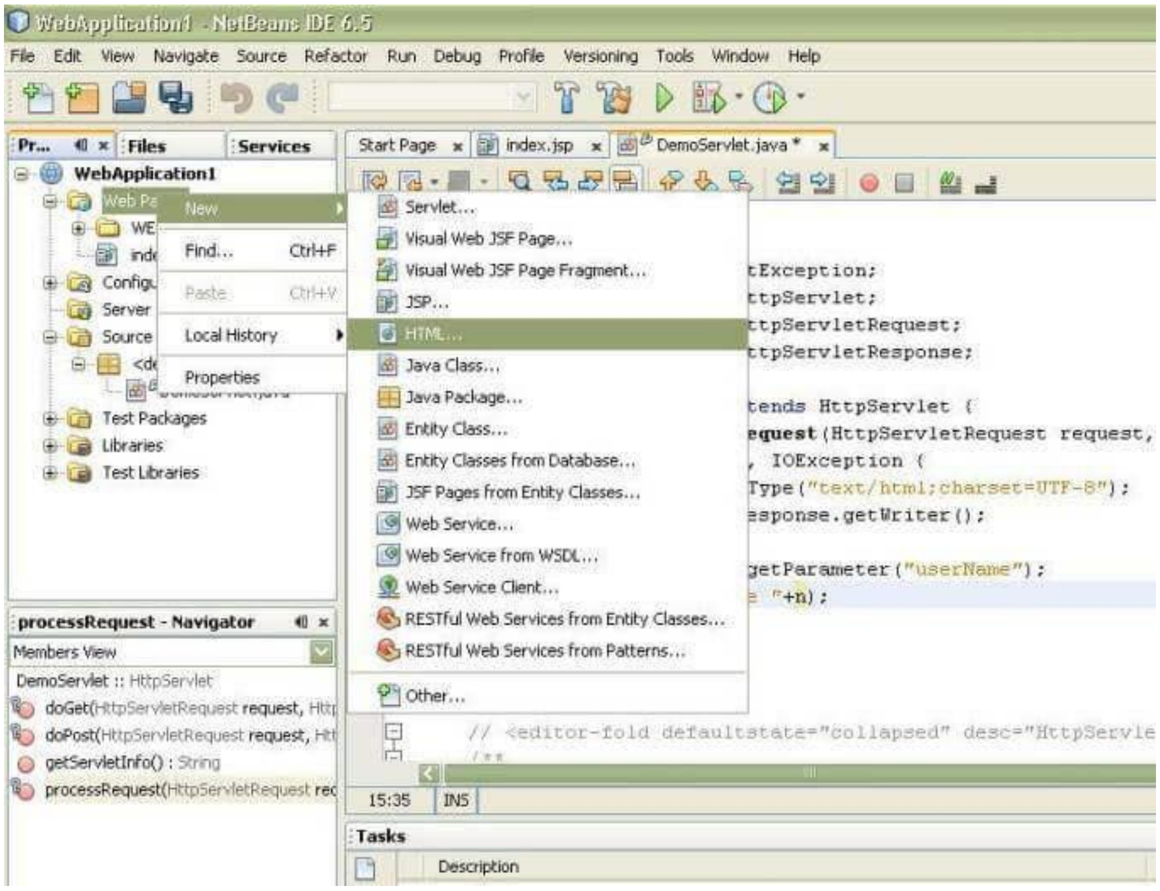
DemoServlet :: HttpServlet
- doGet(HttpServletRequest request, HttpServletR
- doPost(HttpServletRequest request, HttpServletR
- getServletInfo() : String
- processRequest(HttpServletRequest request, Htt

Description	File
/* TODO output your page here	DemoServl...

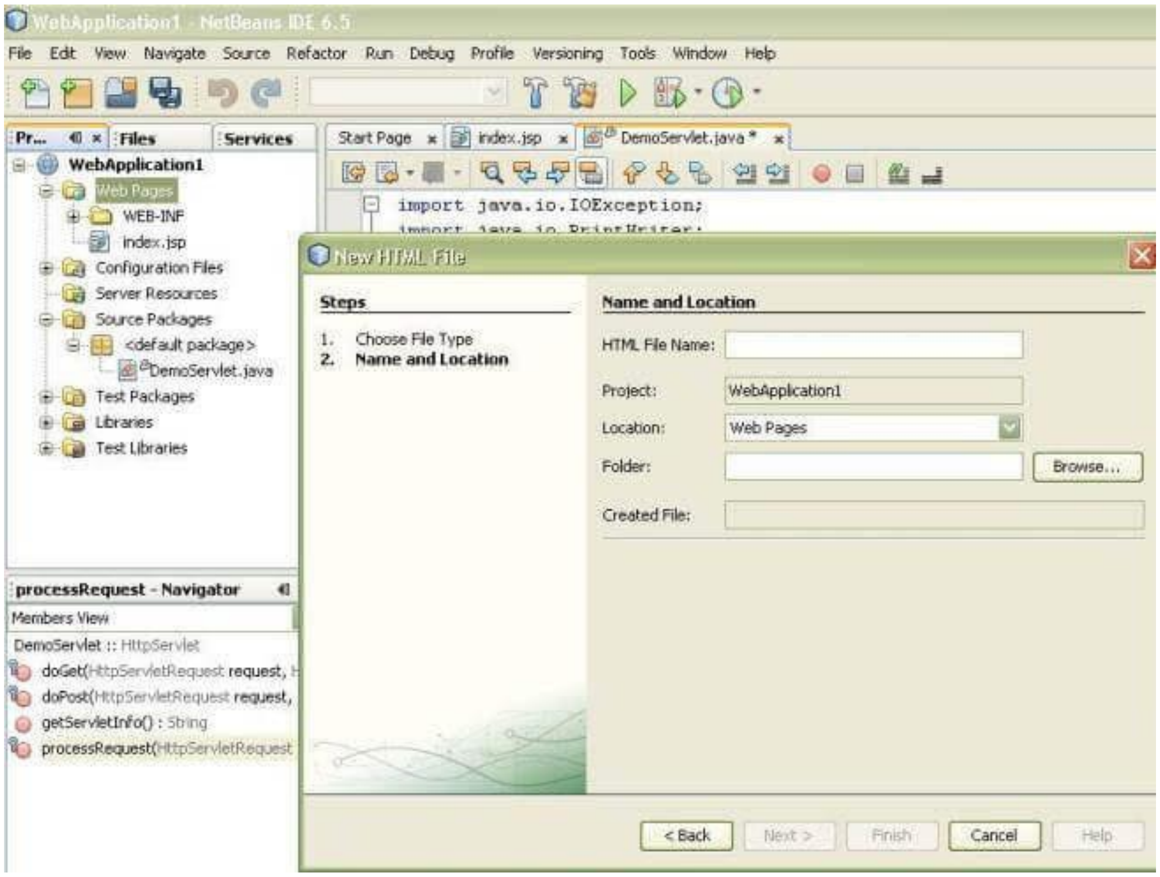
TODO: 1 in all opened projects



```
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;

public class DemoServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpSer
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.get
```


WebApplication1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Pr... Files Services
Start Page index.jsp DemoServlet.java *
WebApplication1
Web Pa
WE
inde
Config
Server
Source
<de
Test Packages
Libraries
Test Libraries
New
Find... Ctrl+F
Paste Ctrl+V
Local History
Properties
Servlet...
Visual Web JSF Page...
Visual Web JSF Page Fragment...
JSP...
HTML...
Java Class...
Java Package...
Entity Class...
Entity Classes from Database...
JSF Pages from Entity Classes...
Web Service...
Web Service from WSDL...
Web Service Client...
RESTful Web Services from Entity Classes...
RESTful Web Services from Patterns...
Other...
processRequest - Navigator
Members View
DemoServlet :: HttpServlet
doGet(HttpServletRequest request, Http
doPost(HttpServletRequest request, Htt
getServletInfo() : String
processRequest(HttpServletRequest req
15:35 INS
Tasks
Description

```
tException;
ttpServlet;
ttpServletRequest;
ttpServletResponse;

tends HttpServlet {
equest(HttpServletRequest request,
, IOException {
Type("text/html;charset=UTF-8");
esponse.getWriter();

getParameter("userName");
= "+n);

    // <editor-fold defaultstate="collapsed" desc="HttpServle
    /**
```


WebApplication1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Pr... Files Services
Start Page index.jsp DemoServlet.java *
WebApplication1
Web Pages
WEB-INF
index.jsp
Configuration Files
Server Resources
Source Packages
<default package>
DemoServlet.java
Test Packages
Libraries
Test Libraries
processRequest - Navigator
Members View
DemoServlet :: HttpServlet
doGet(HttpServletRequest request,
doPost(HttpServletRequest request,
getServletInfo() : String
processRequest(HttpServletRequest
New HTML File
Steps
1. Choose File Type
2. Name and Location
Name and Location
Browse...
< Back
Next >
Finish
Cancel
Help

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

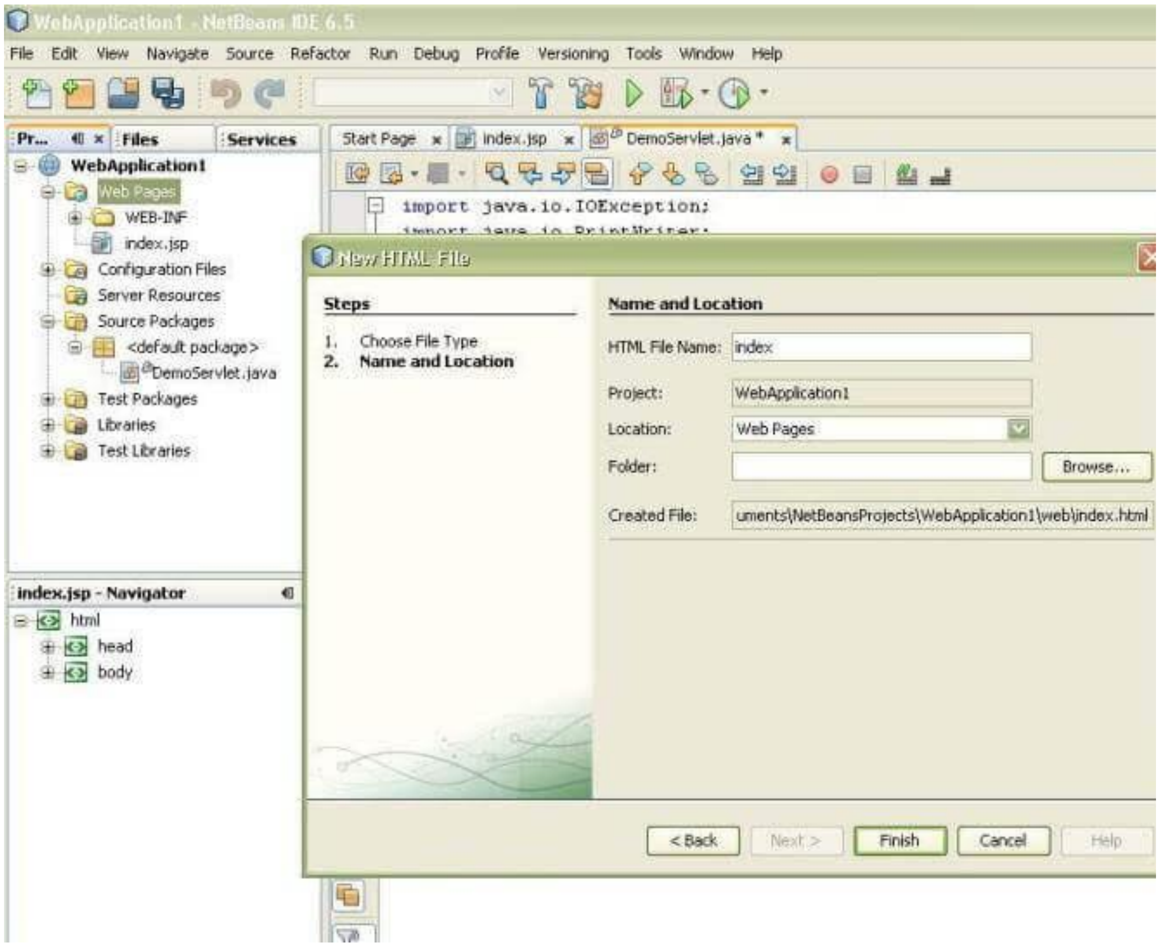
HTML File Name:

Project: WebApplication1

Location: Web Pages

Folder:

Created File:



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

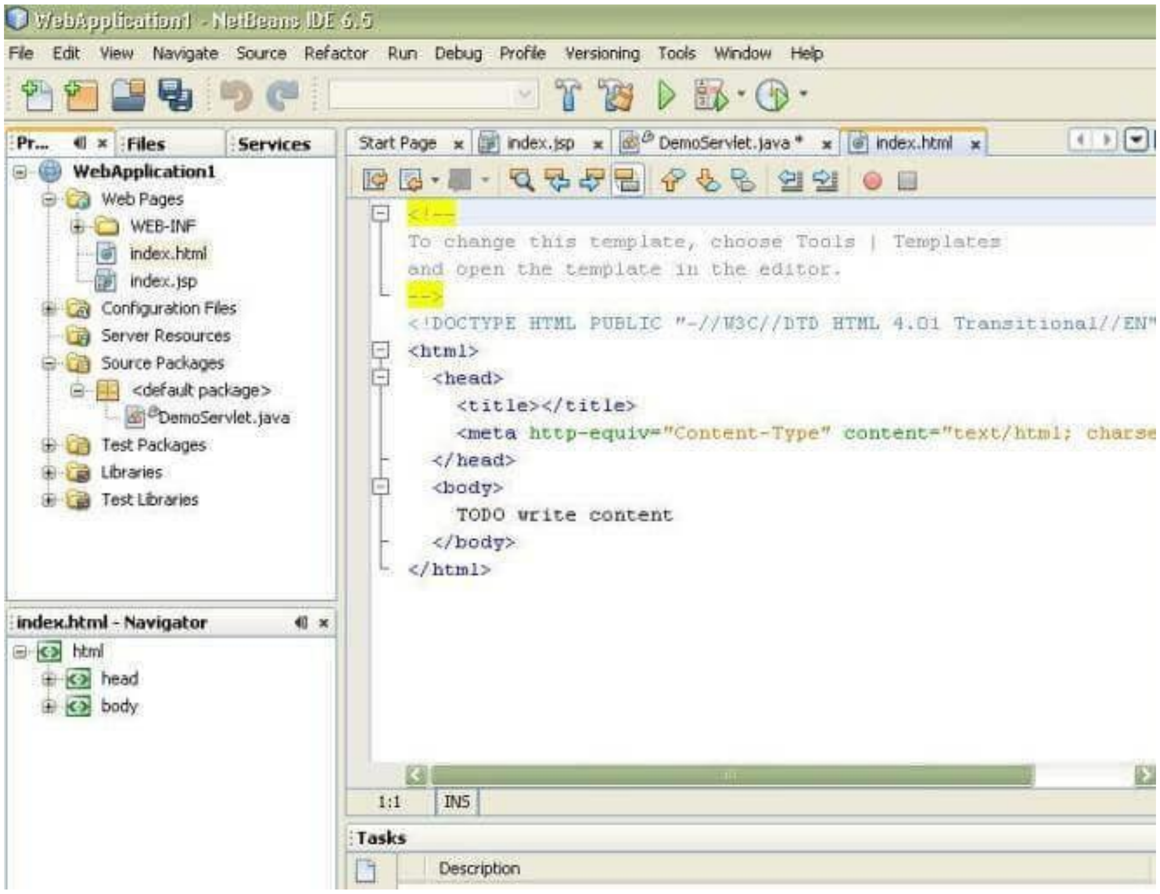
HTML File Name: index

Project: WebApplication1

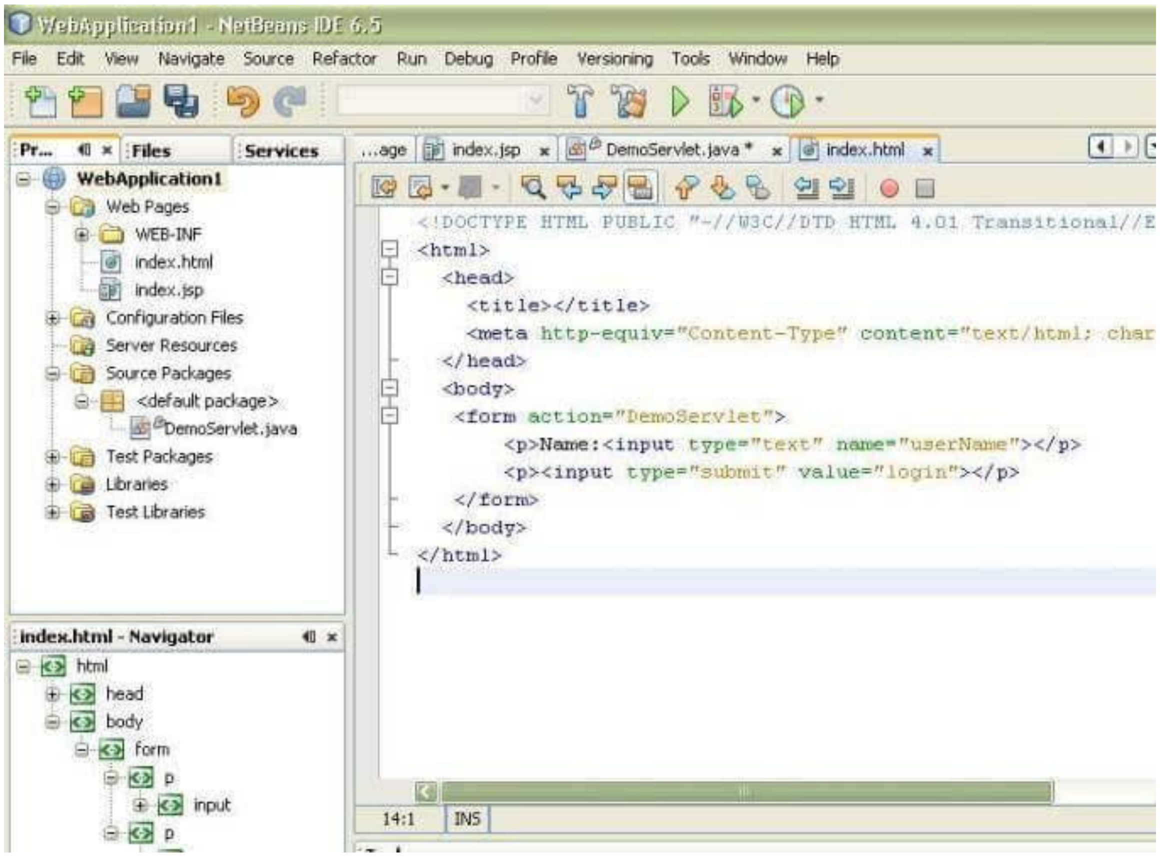
Location: Web Pages

Folder:

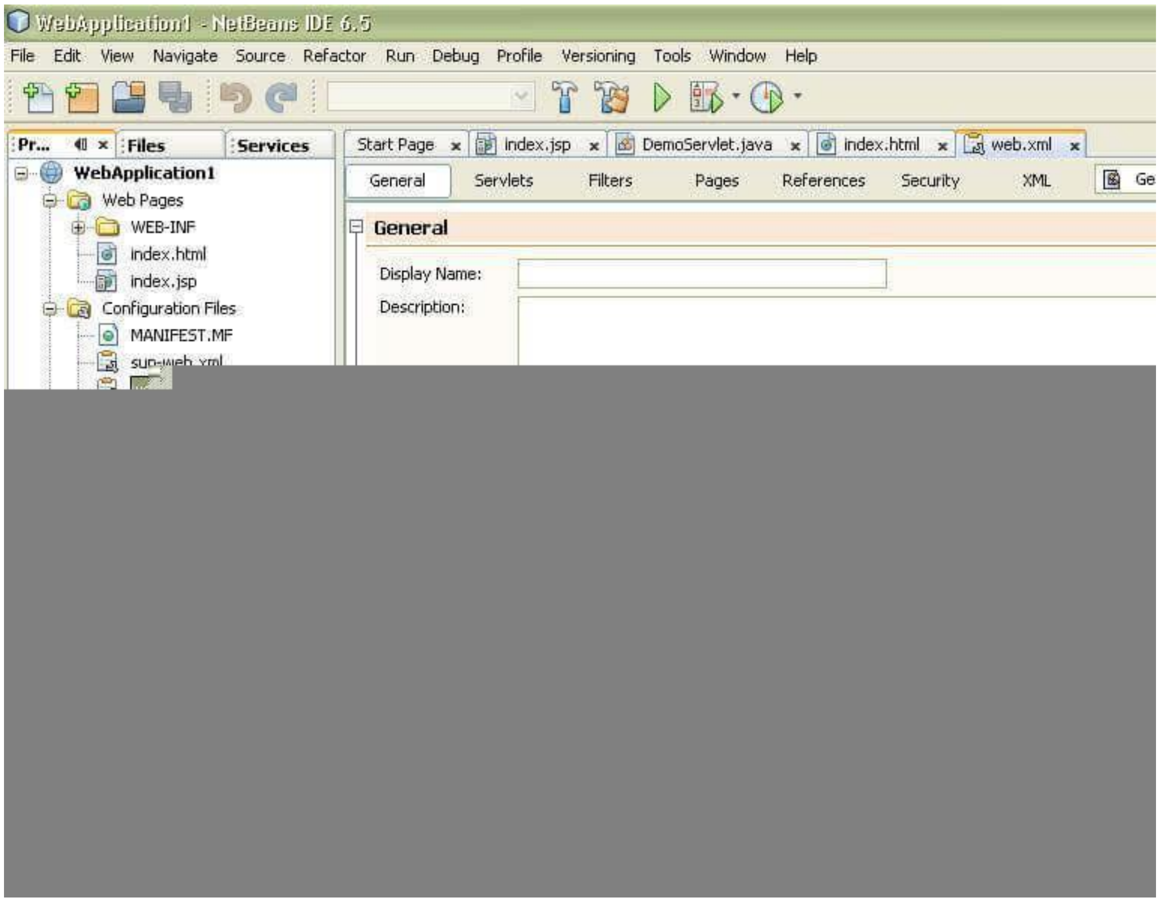
Created File: uments\NetBeansProjects\WebApplication1\web\index.html

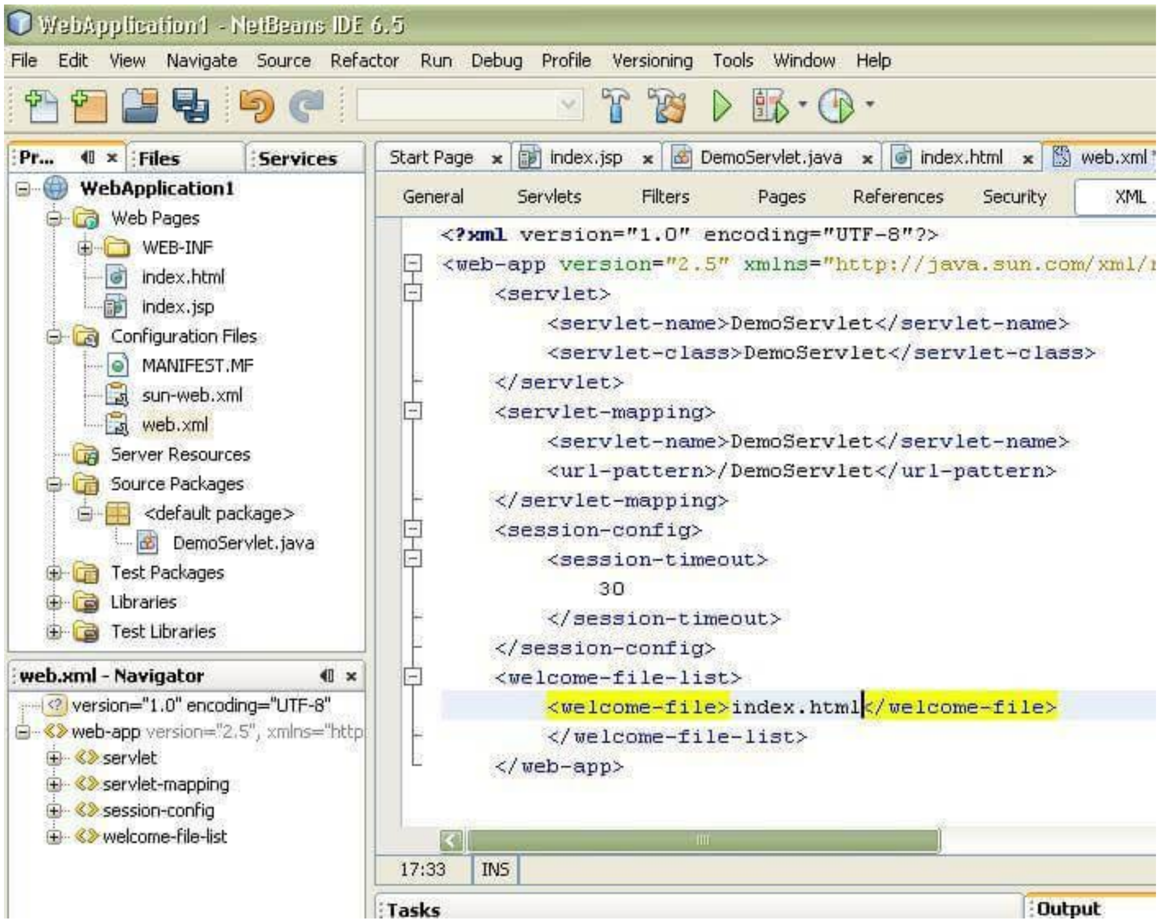


```
<!--
To change this template, choose Tools | Templates
and open the template in the editor.
-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; charse
  </head>
  <body>
    TODO write content
  </body>
</html>
```

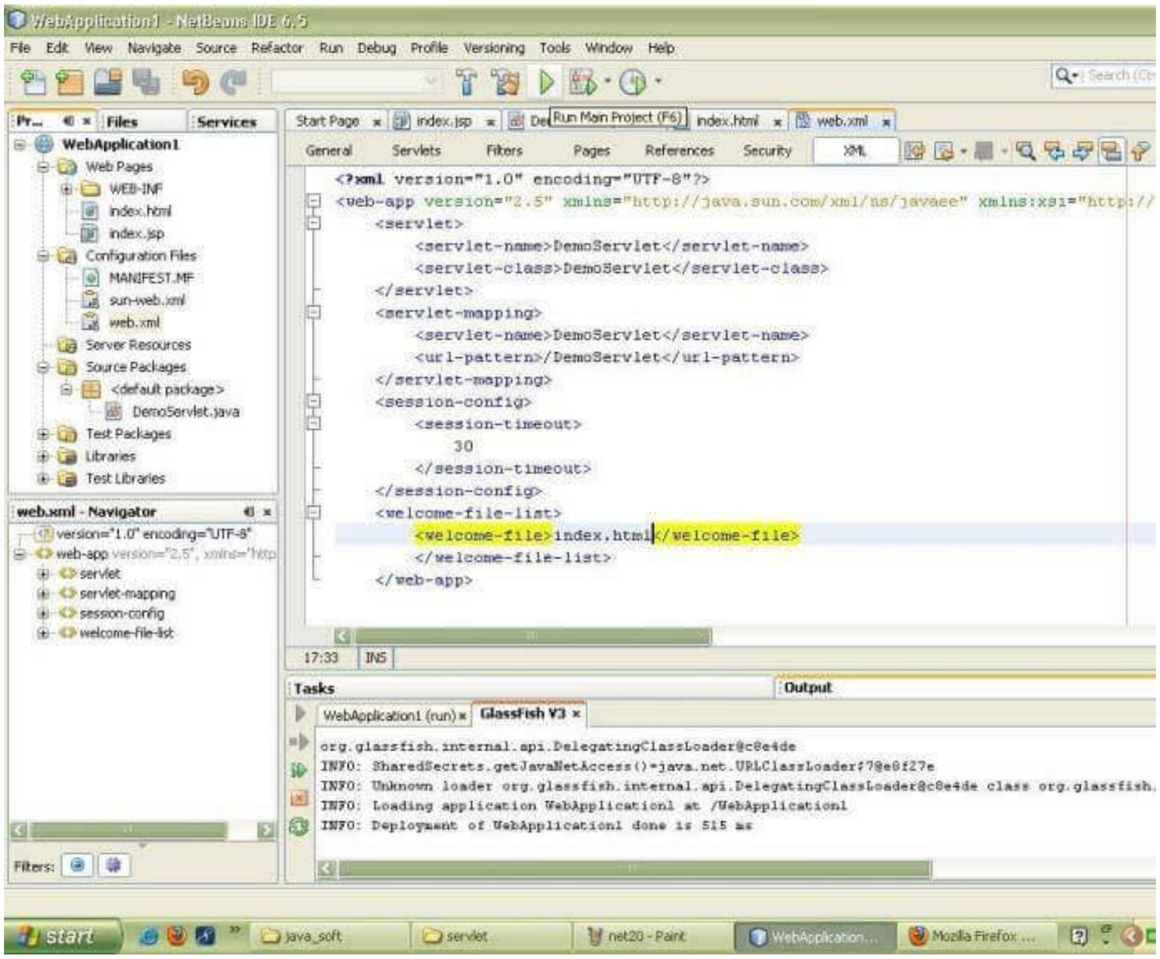


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//E
<html>
  <head>
    <title></title>
    <meta http-equiv="Content-Type" content="text/html; char
  </head>
  <body>
   <form action="DemoServlet">
       <p>Name:<input type="text" name="userName"></p>
       <p><input type="submit" value="login"></p>
   </form>
  </body>
</html>
```


WebApplication1 - NetBeans IDE 6.5
File Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Pr... Files Services
WebApplication1
Web Pages
WEB-INF
index.html
index.jsp
Configuration Files
MANIFEST.MF
sun-web.xml
Start Page index.jsp DemoServlet.java index.html web.xml
General Servlets Filters Pages References Security XML
General
Display Name:
Description:

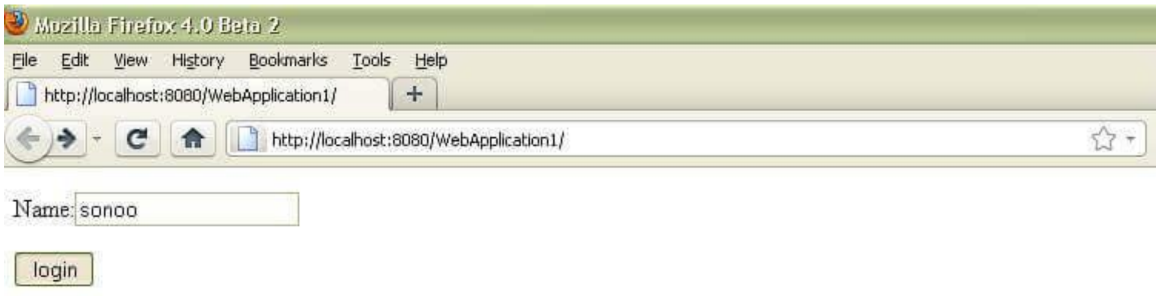


```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/
    <servlet>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DemoServlet</servlet-name>
        <url-pattern>/DemoServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        </welcome-file-list>
    </web-app>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://
    <servlet>
        <servlet-name>DemoServlet</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>DemoServlet</servlet-name>
        <url-pattern>/DemoServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        </welcome-file-list>
    </web-app>
```

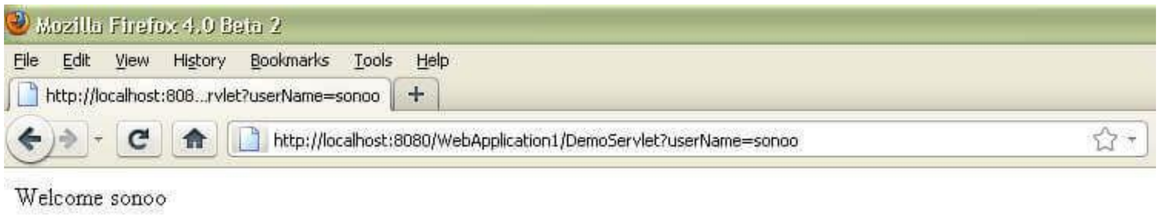
```
org.glassfish.internal.api.DelegatingClassLoader@c8e4de
INFO: SharedSecrets.getJavaNetAccess()=java.net.URLClassLoader$7@e8f27e
INFO: Unknown loader org.glassfish.internal.api.DelegatingClassLoader@c8e4de class org.glassfish.
INFO: Loading application WebApplication1 at /WebApplication1
INFO: Deployment of WebApplication1 done is 515 ms
```



http://localhost:8080/WebApplication1/

Name: sonoo

login


Mozilla Firefox 4.0 Beta 2
File Edit View History Bookmarks Tools Help
http://localhost:808...rvlet?userName=sonoo

```
http://localhost:8080/WebApplication1/DemoServlet?userName=sonoo
```

```
Welcome sonoo
```