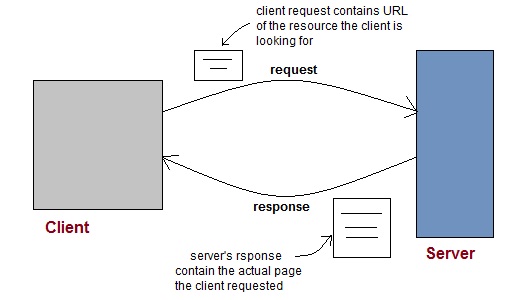
**Introduction to Web**

Web consists of billions of clients and server connected through wires and wireless networks. The web clients make requests to web server. The web server receives the request, finds the resources and return the response to the client. When a server answers a request, it usually sends some type of content to the client. The client uses web browser to send request to the server. The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language). All browsers know how to display HTML page to the client.

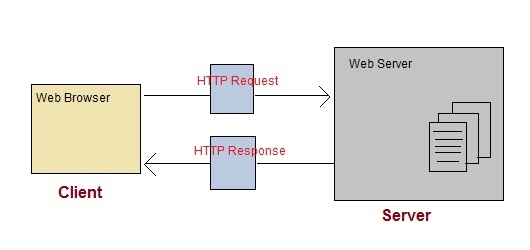


**Web Application**

A website is a collection of static files(webpages) such as HTML pages, images, graphics etc. A **Web application** is a web site with dynamic functionality on the server. **Google**, **Facebook**, **Twitter** are examples of web applications.

**HTTP (Hypertext Transfer Protocol)**

* HTTP is a protocol that clients and servers use on the web to communicate.
* It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
* HTTP is a **stateless protocol** i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
* The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.



**HTTP Methods**

HTTP request can be made using a variety of methods, but the ones you will use most often are **Get** and **Post**. The method name tells the server the kind of request that is being made, and how the rest of the message will be formated.

**HTTP Methods and Descriptions :**

| **Method Name** | **Description** |
| --- | --- |
| OPTIONS | Request for communication options that are available on the request/response chain. |
| GET | Request to retrieve information from server using a given URI. |
| HEAD | Identical to GET except that it does not return a message-body, only the headers and status line. |
| POST | Request for server to accept the entity enclosed in the body of HTTP method. |
| DELETE | Request for the Server to delete the resource. |
| CONNECT | Reserved for use with a proxy that can switch to being a tunnel. |
| PUT | This is same as POST, but POST is used to create, PUT can be used to create as well as update. It replaces all current representations of the target resource with the uploaded content. |

**Difference between GET and POST requests**

| **GET Request** | **POST Request** |
| --- | --- |
| Data is sent in header to the server | Data is sent in the request body |
| Get request can send only limited amount of data | Large amount of data can be sent. |
| Get request is not secured because data is exposed in URL | Post request is secured because data is not exposed in URL. |
| Get request can be bookmarked and is more efficient. | Post request cannot be bookmarked. |

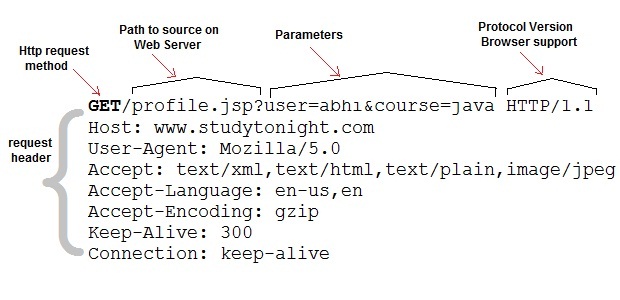
**General Difference between PUT and POST methods**

Following are some basic differences between the PUT and the POST methods :

* **POST** to a URL creates a child resource at a server defined URL while **PUT** to a URL creates/replaces the resource in its entirety at the client defined URL.
* POST creates a child resource, so POST to /books will create a resources that will live under the /booksresource. Eg. /books/1. Sending the same post request twice will create two resources.
* PUT is for creating or replacing a resource at a URL known by the client.
* PUT must be used for CREATE when the client already knows the url before the resource is created.
* PUT replaces the resource at the known url if it already exists, so sending the same request twice has no effect. In other words, calls to PUT are **idempotent.**

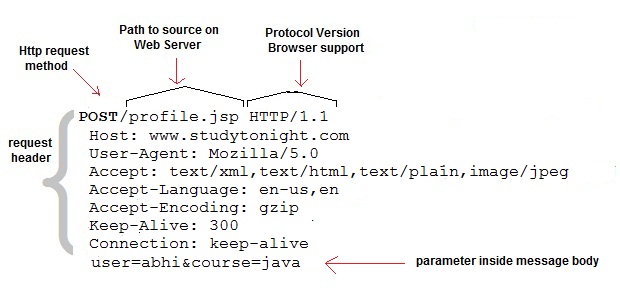
**Anatomy of an HTTP GET request**

Get request contains path to server and the parameters added to it.



**Anatomy of an HTTP POST request**

Post requests are used to make more complex requests on the server. For instance, if a user has filled a form with multiple fields and the application wants to save all the form data to the database. Then the form data will be sent to the server in POST request body, which is also known as Message body.



**Steps to Create Servlet Application using tomcat server**

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of [Tomcat Server](http://tomcat.apache.org/) and install it on your machine.

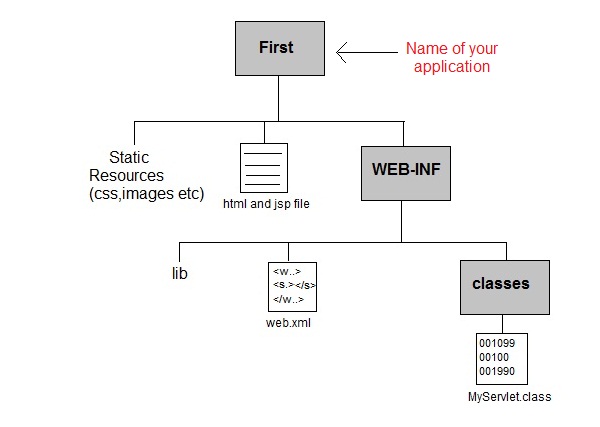
After installing Tomcat Server on your machine follow the below mentioned steps :

1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployement Descriptor for your application
5. Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

**1. Creating the Directory Structure**

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, css etc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside classesfolder.

The web.xml (deployement descriptor) file is kept under WEB-INF folder.

**Creating a Servlet**

There are three different ways to create a servlet.

* By implementing **Servlet** interface
* By extending **GenericServlet** class
* By extending **HttpServlet** class

But mostly a servlet is created by extending **HttpServlet** abstract class. As discussed earlier **HttpServlet** gives the definition of service() method of the **Servlet** interface. The servlet class that we will create should not override service() method. Our servlet class will override only doGet() or doPost() method.

When a request comes in for the servlet, the Web Container calls the servlet's service() method and depending on the type of request the service() method calls either the doGet() or doPost() method.

**NOTE:** By default a request is **Get** request.

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

public *MyServlet* **extends** HttpServlet

{

public void *doGet*(HttpServletRequest request,HttpServletResposne response)

**throws** ServletException

{

response.setContentType("text/html");

PrintWriter *out* = response.getWriter();

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

out.println("<h1>Hello Readers</h1>");

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

}

}

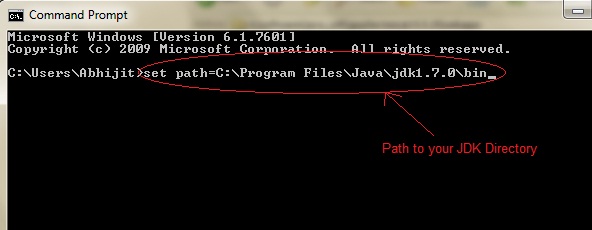
Write above code in a notepad file and save it as **MyServlet.java** anywhere on your PC. Compile it(explained in next step) from there and paste the class file into WEB-INF/classes/ directory that you have to create inside **Tomcat/webapps** directory.

**Compiling a Servlet**

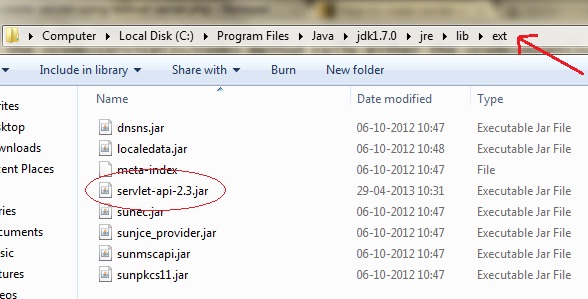
To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

Steps to compile a Servlet

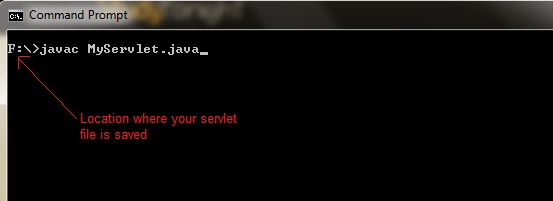
* Set the Class Path.



* Download **servlet-api.jar** file.
* Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.



* Compile the Servlet class.



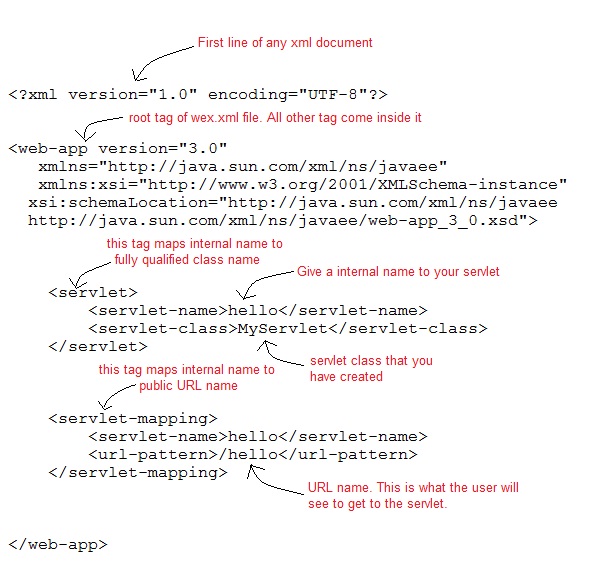
**NOTE:**After compiling your Servlet class you will have to paste the class file into WEB-INF/classes/ directory.

**Create Deployment Descriptor**

**Deployment Descriptor(DD)** is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

* Mapping URL to Servlet class.
* Initializing parameters.
* Defining Error page.
* Security roles.
* Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.



**Start the Server**

Double click on the **startup.bat** file to start your Apache Tomcat Server.

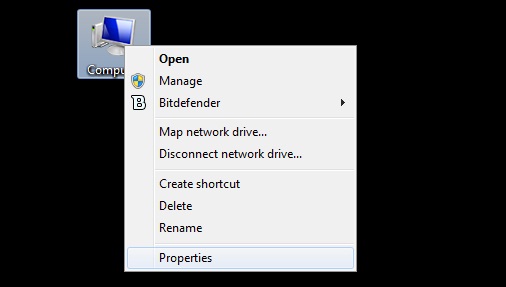
Or, execute the following command on your windows machine using RUN prompt.

C:\apache-tomcat-7.0.14\bin\startup.bat

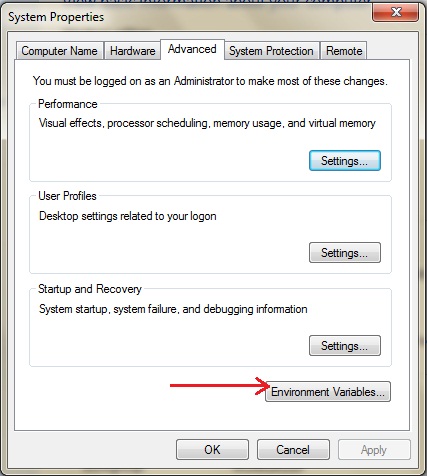
**Starting Tomcat Server for the first time**

If you are starting Tomcat Server for the first time you need to set JAVA\_HOME in the Enviroment variable. The following steps will show you how to set it.

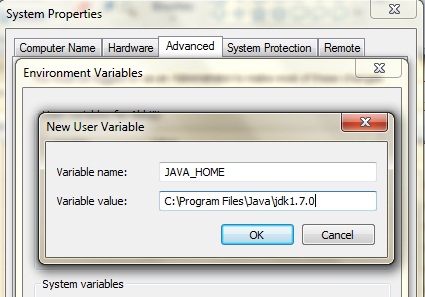
* Right Click on **My Computer**, go to **Properites**.



* Go to **Advanced** Tab and Click on **Enviroment Variables...** button.

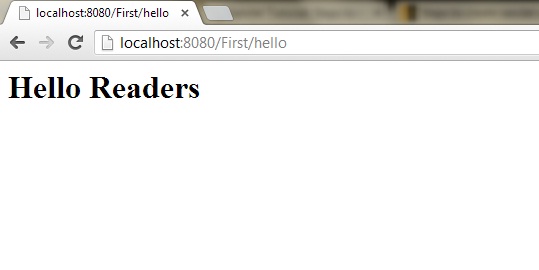


* Click on **New** button, and enter **JAVA\_HOME** inside Variable name text field and path of JDK inside Variable value text field. Click OK to save.



**Run Servlet Application**

Open Browser and type **http:localhost:8080/First/hello**



Hurray! Our first Servlet Application ran successfully.

**Creating First Servlet Application using Netbeans IDE**

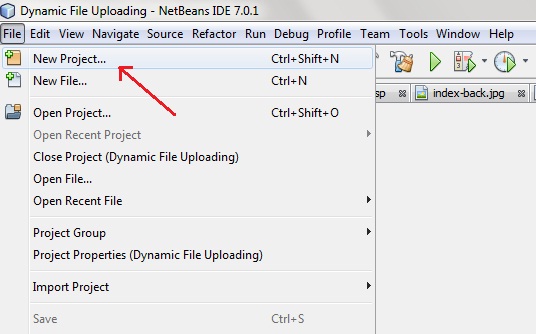
In the last lesson we created our first Servlet Application but without using any IDE. An IDE is Integrated Development Environment, and it makes creating applications a lot easier. We will learn how to create Servlet applications on NetBeans IDE and Eclipse IDE. Then you can decide which one, you want to use.

Using Intregrated Development Enviroment(IDE) is the easiest way to create Servlet Applications. An IDE is a software application that provides facilities to computer programmers for software development. **Eclipse**, **MyEcplise**, **Netbeans** are example of some popular Java IDE.

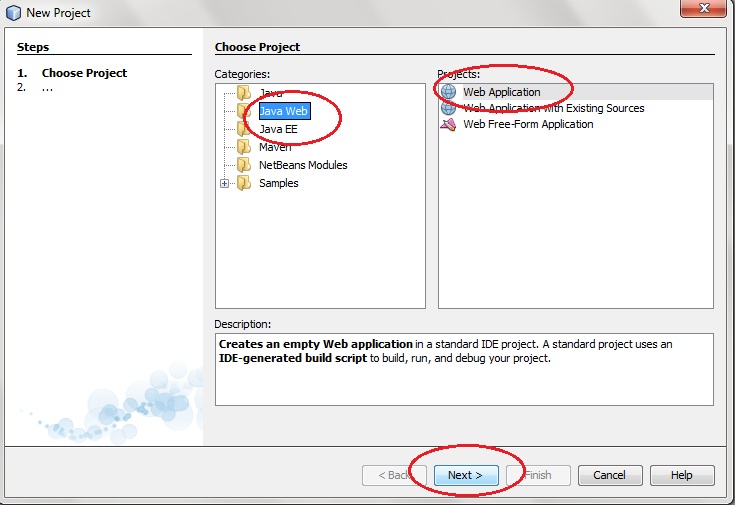
**Steps to Create Servlet Application in Netbeans IDE**

To create a servlet application in Netbeans IDE, you will need to follow the following (simple) steps :

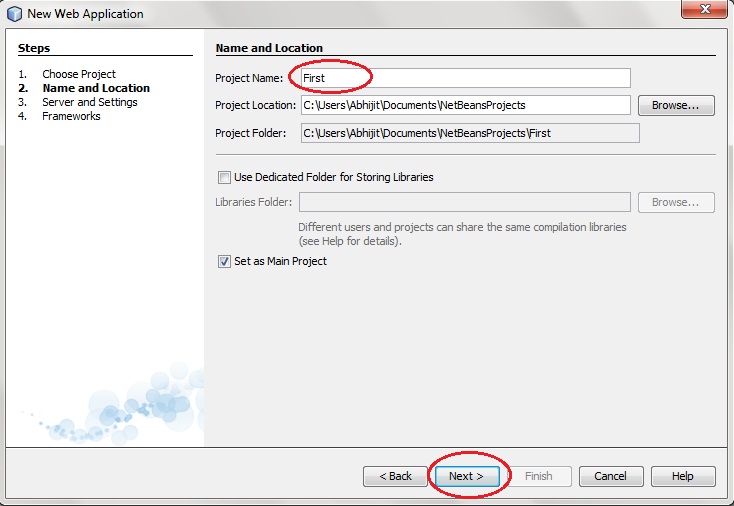
1. Open Netbeans IDE, Select **File** -> **New Project**



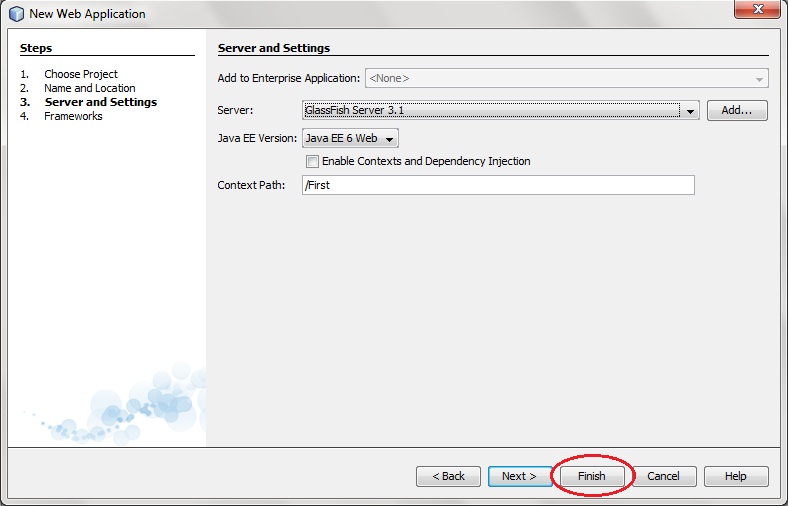
1. Select **Java Web** -> **Web Application**, then click on Next,



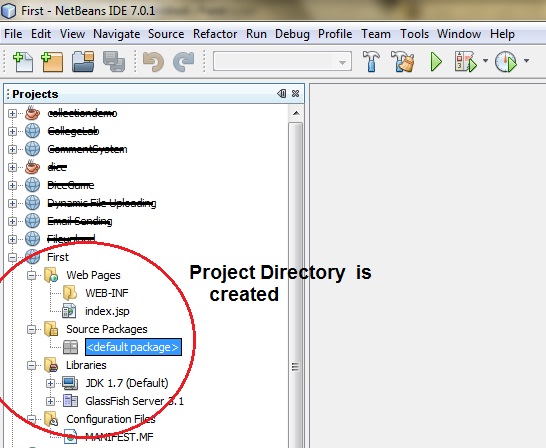
1. Give a name to your project and click on Next,



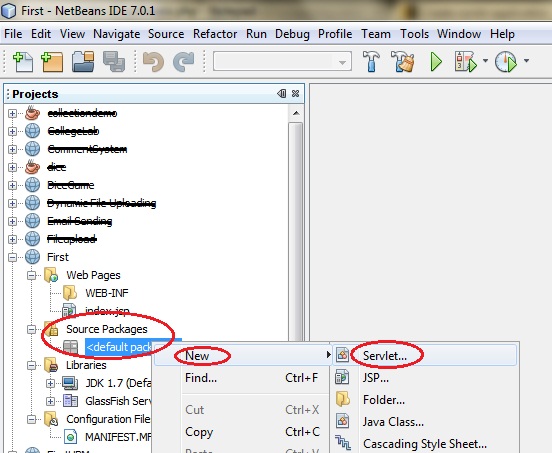
1. and then, Click **Finish**



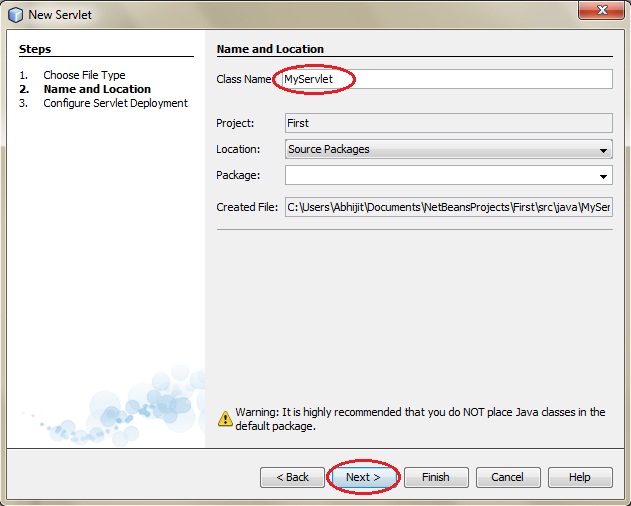
1. The complete directory structure required for the Servlet Application will be created automatically by the IDE.

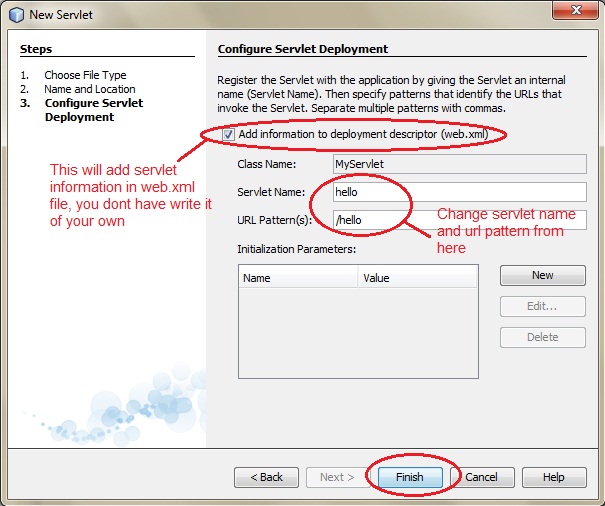


1. To create a Servlet, open **Source Package**, right click on **default packages** -> **New** -> **Servlet**.

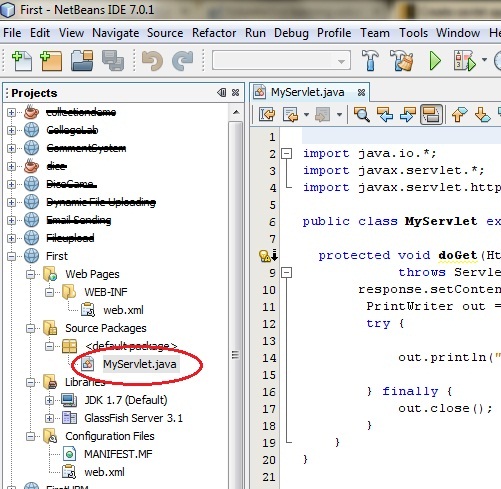


1. Give a Name to your Servlet class file,

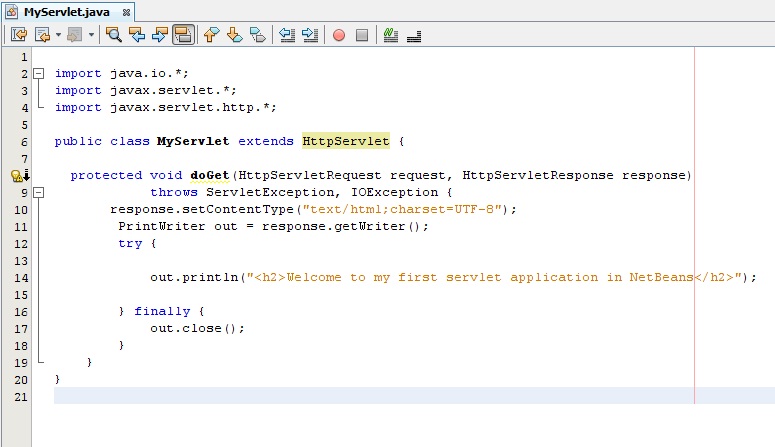




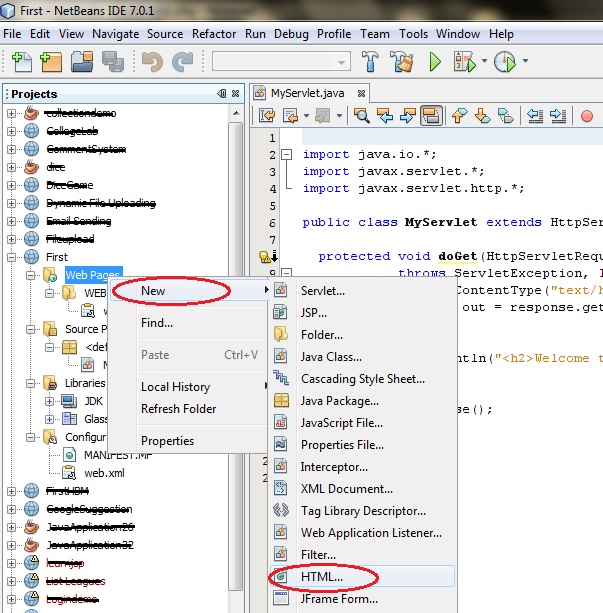
1. Now, your Servlet class is ready, and you just need to change the method definitions and you will good to go.



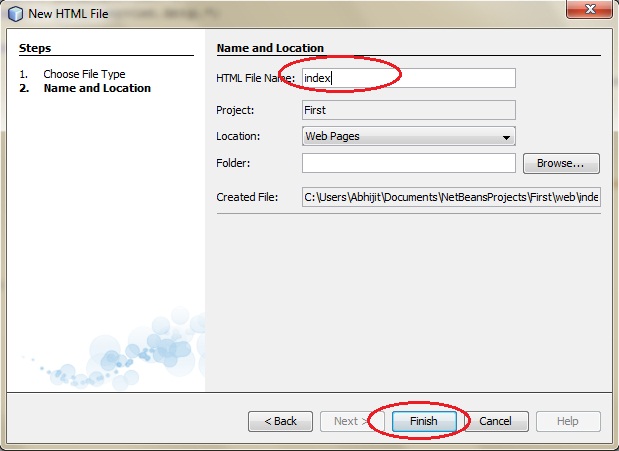
1. Write some code inside your Servlet class.



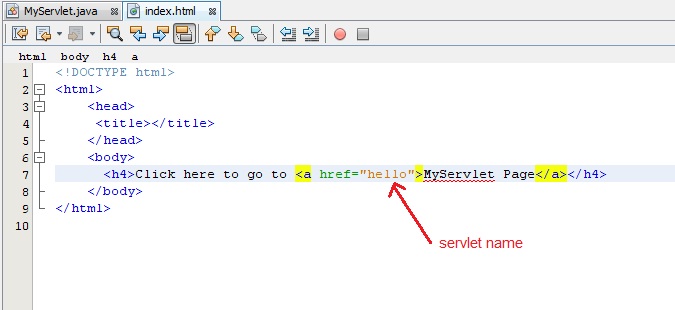
1. Create an HTML file, right click on **Web Pages** -> **New** -> **HTML**



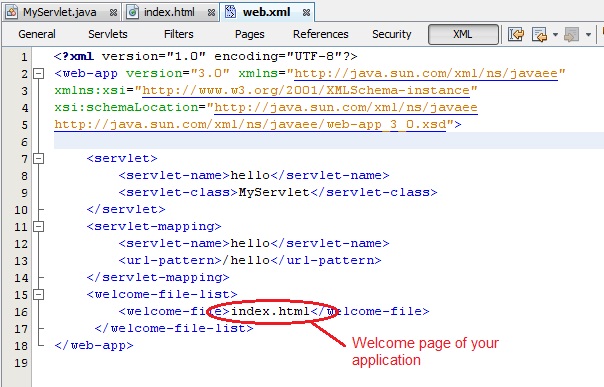
1. Give it a name. We recommend you to name it index, because browser will always pick up the index.html file automatically from a directory. Index file is read as the first page of the web application.



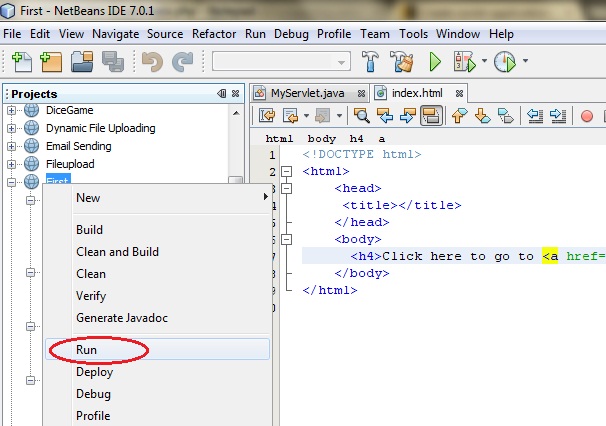
1. Write some code inside your HTML file. We have created a hyperlink to our Servlet in our HTML file.



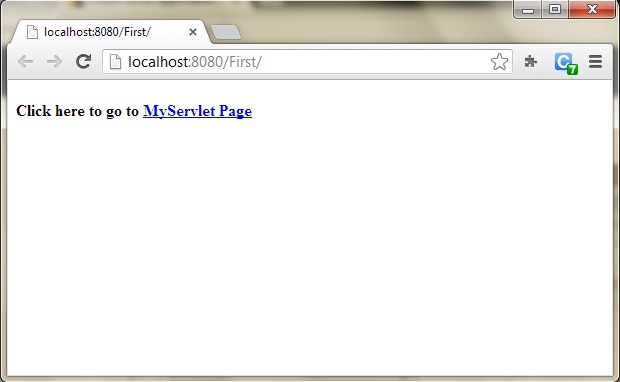
1. Edit **web.xml** file. In the web.xml file you can see, we have specified the **url-pattern** and the **servlet-name**, this means when hello url is accessed our Servlet file will be executed.



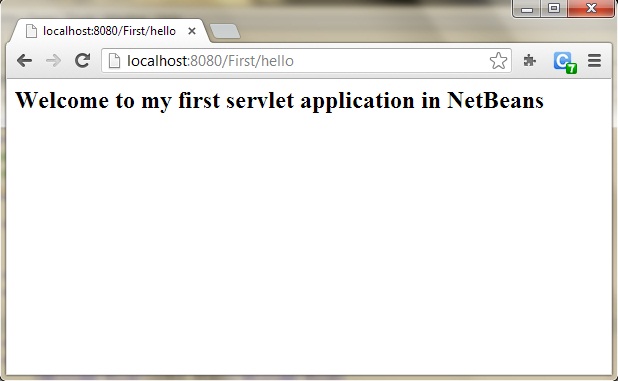
1. Run your application, right click on your Project and select **Run**



1. Click on the link created, to open your Servlet.



1. Hurray! Our First Servlet class is running.



### Introduction to Servlet Request

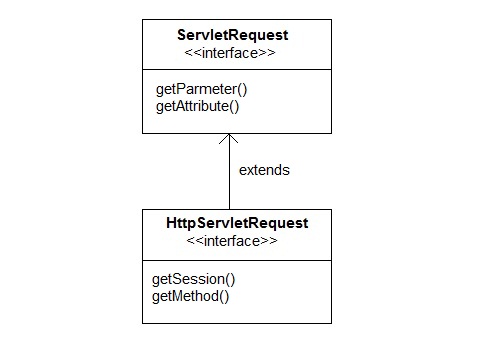
True job of a Servlet is to handle client request. Servlet API provides two important interfaces **javax.servlet.ServletRequest** and **javax.servlet.http.HttpServletRequest** to encapsulate client request. Implementation of these interfaces provide important information about client request to a servlet.

#### Some Important Methods of ServletRequest

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Object getAttribute(String name) | return attribute set on request object by name |
| Enumeration getAttributeName() | return an Enumeration containing the names of the attributes available inthis request |
| int getContentLength() | return size of request body |
| int getContentType() | return media type of request content |
| ServletInputStream getInputStream() | returns a input stream for reading binary data |
| String getParameter(String name) | returns value of parameter by name |
| String getLocalAddr() | returns the Internet Protocol(IP) address of the interface on which the request was received |
| Enumeration getParameterNames() | returns an enumeration of all parameter names |
| String[] getParameterValues(String name) | returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist |
| ServletContext getServletContext() | return the servlet context of current request. |
| String getServerName() | returns the host name of the server to which the request was sent |
| int getServerPort() | returns the port number to which the request was sent |
| boolean isSecure() | returns a boolean indicating whether this request was made using a secure channel, such as HTTPS. |
| void removeAttribute(String name) | removes an attribute from this request |
| void setAttribute(String name, Object o) | stores an attribute in this request. |

#### HttpServletRequest interface

**HttpServletRequest** interface adds the methods that relates to the **HTTP** protocol.



#### Some important methods of HttpServletRequest

|  |  |
| --- | --- |
| **Methods** | **Description** |
| String getContextPath() | returns the portion of the request URI that indicates the context of the request |
| Cookies getCookies() | returns an array containing all of the Cookie objects the client sent with this request |
| String getQueryString() | returns the query string that is contained in the request URL after the path |
| HttpSession getSession() | returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session |
| String getMethod() | Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT. |
| Part getPart(String name) | gets the Part with the given name |
| String getPathInfo() | returns any extra path information associated with the URL the client sent when it made this request. |
| String getServletPath() | returns the part of this request's URL that calls the servlet |

#### Example demonstrating Servlet Request

In this example, we will show how a parameter is passed to a Servlet in a request object from HTML page.

**index.html**

<form method="post" action="check">

Name <input type="text" name="user" >

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

</form>

**web.xml**

<servlet>

<servlet-name>check</servlet-name>

<servlet-class>MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>check</servlet-name>

<url-pattern>/check</url-pattern>

</servlet-mapping>

**MyServlet.java**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class *MyServlet* **extends** HttpServlet {

protected void *doPost*(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter *out* = response.getWriter();

try {

String *user*=request.getParameter("user");

out.println("<h2> Welcome "+user+"</h2>");

} finally {

out.close();

}

}

}

**Output :**

