

JavaBean

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Simple example of JavaBean class

```
1. //Employee.java
2.
3. package mypack;
4. public class Employee implements java.io.Serializable{
5. private int id;
6. private String name;
7. public Employee(){
8. public void setId(int id){this.id=id;}
9. public int getId(){return id;}
10. public void setName(String name){this.name=name;}
11. public String getName(){return name;}
12. }
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods.

```
package mypack;
```

```
public class Test{
```

```
public static void main(String args[]){  
  
Employee e=new Employee();//object is created  
  
e.setName("Arjun");//setting value to the object  
  
System.out.println(e.getName());  
  
}}
```

Note: There are two ways to provide values to the object. One way is by constructor and second is by setter method.

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. **getPropertyName ()**

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. **setPropertyName ()**

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean:

- The JavaBean properties and methods can be exposed to another application.
- It provides an easiness to reuse the software components.

Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

- JavaBeans are mutable. So, it can't take advantages of immutable objects.
- Creating the setter and getter method for each property separately may lead to the boilerplate code.

Servlet: 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.

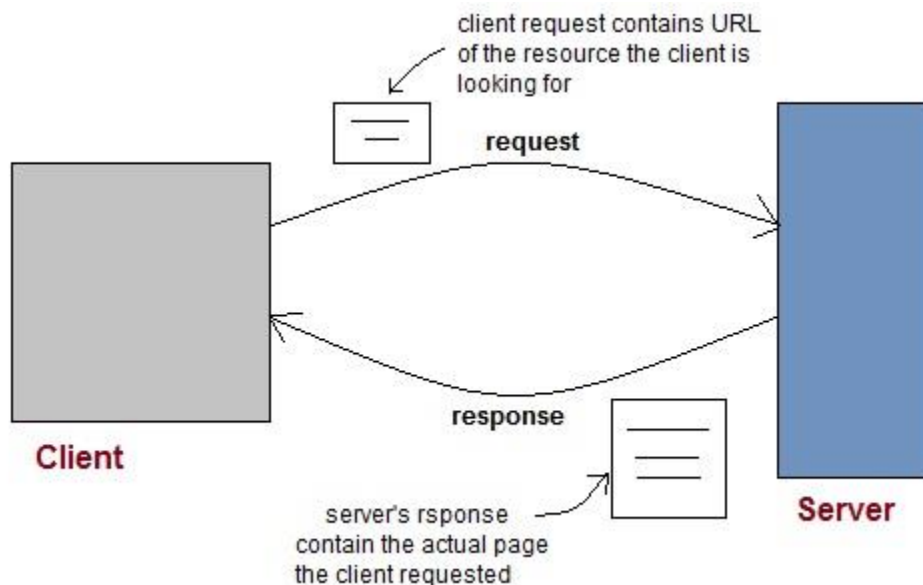
The web server receives the request, finds the resources and return the response to the client.

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(Hyper Text Markup Language).

All browsers know how to display HTML page to the client.



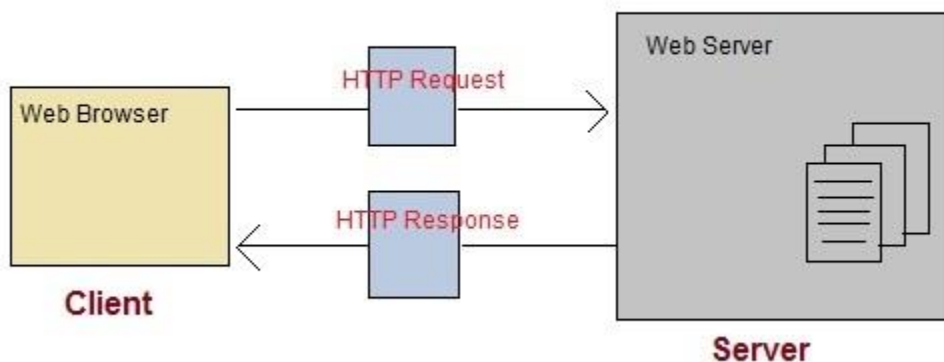
Web Application

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 formatted

.

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

Method Name	Description
	uploaded content.

Difference between GET and POST requests

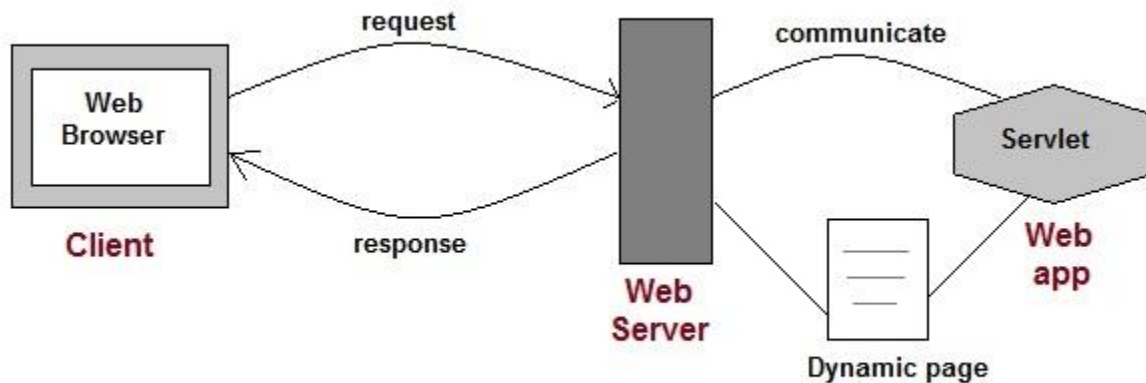
GET Request	POST Request
Data is sent in header to the server	Data is sent in the request body
Get request can send only limited amount of data	Large amount of data can be sent.
Get request is not secured because data is exposed in URL	Post request is secured because data is not exposed in URL.
Get request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

Servlet

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

Servlet Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.



As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

Execution of Servlets :

Execution of Servlets involves six basic steps:

1. The clients send the request to the web server.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the web server.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Difference between Servlet and CGI

Servlet

Servlets are portable and efficient.

In Servlets, sharing of data is possible.

Servlets can directly communicate with the web server.

Servlets are less expensive than CGI.

Servlets can handle the cookies.

CGI(Common Gateway Interface)

CGI is not portable

In CGI, sharing of data is not possible.

CGI cannot directly communicate with the web server.

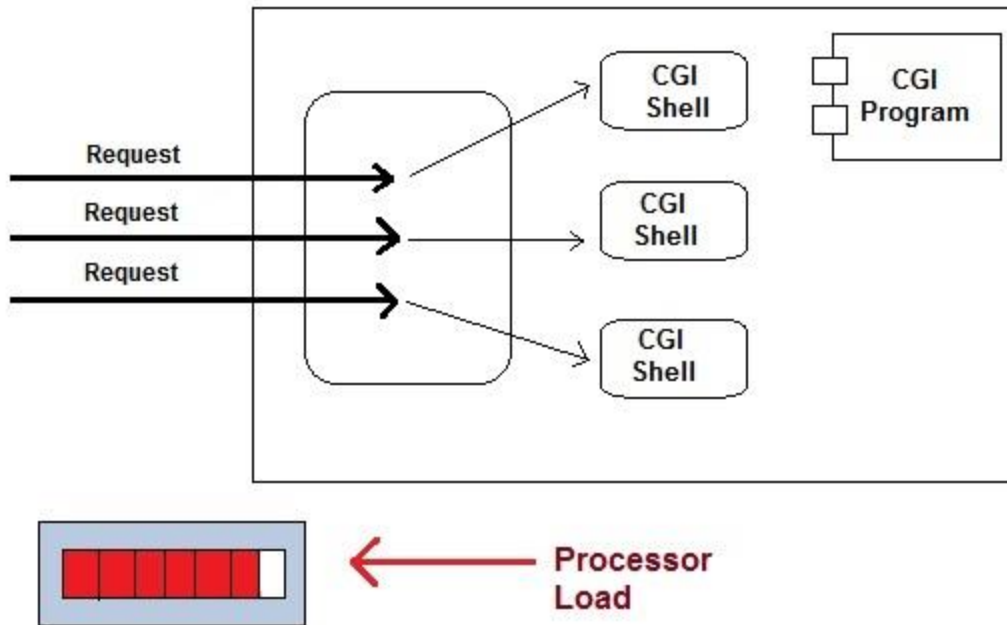
CGI are more expensive than Servlets.

CGI cannot handle the cookies.

CGI (Common Gateway Interface)

Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works :

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



Drawbacks of CGI programs

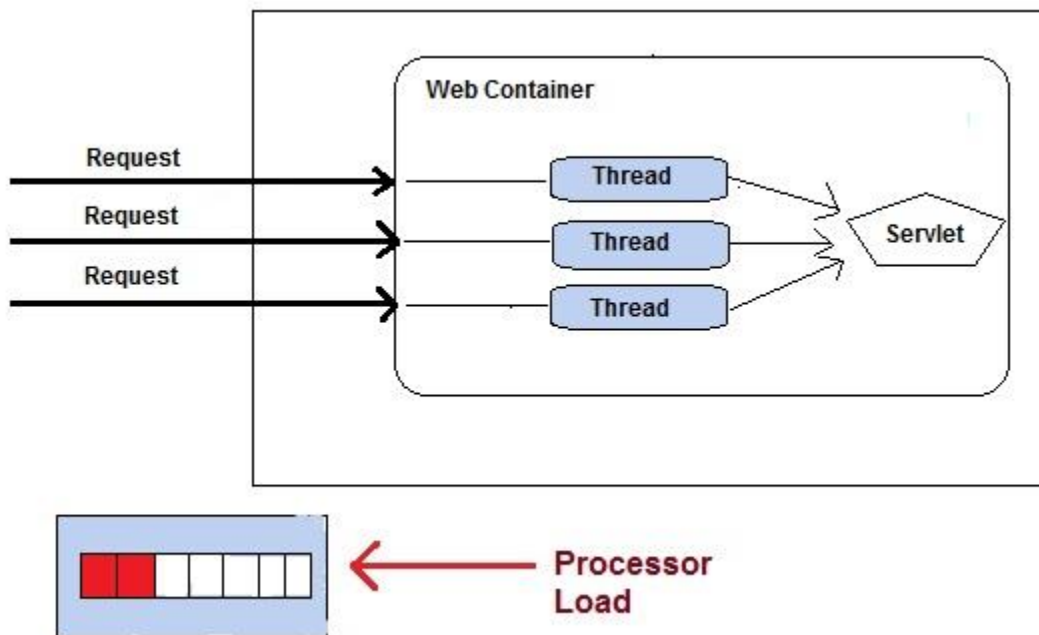
- High response time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Advantages of using Servlets

- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.

- Servlets are platform independent.



Note:- The **javax** prefix is used by the Java programming language for a package of standard Java extensions. These include extensions such as **javax.servlet**, which deals with running servlets, and **javax.jcr**, which deals with the Java Content Library.

What is a web container?

A **web container** is the component of a web server that interacts with Java servlets. A web container manages the life cycle of servlets; it maps a URL to a particular servlet while ensuring that the requester has relevant access-rights.

Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- `javax.servlet`
- `javax.servlet.http`

Some Important Classes and Interfaces of `javax.servlet`

INTERFACES	CLASSES
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletRequestWrapper

ServletRequest	ServletResponseWrapper
ServletResponse	ServletRequestEvent
ServletContextListener	ServletContextEvent
RequestDispatcher	ServletRequestAttributeEvent
SingleThreadModel	ServletContextAttributeEvent
Filter	ServletException
FilterConfig	UnavailableException
FilterChain	GenericServlet
ServletRequestListener	

Some Important Classes and Interface of `javax.servlet.http`

CLASSES and INTERFACES	
HttpServlet	HttpServletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

Servlet Interface

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.

The diagram illustrates the classification of Servlet Interface methods. It lists five methods: `service(ServletRequest, ServletResponse)`, `init(ServletConfig)`, `destroy()`, `getServletConfig()`, and `getServletInfo()`. Arrows point from the text 'Life Cycle method' to the first three methods (`service`, `init`, and `destroy`). Arrows point from the text 'Non Life Cycle method' to the last two methods (`getServletConfig` and `getServletInfo`).

```
service(ServletRequest, ServletResponse)
init(ServletConfig)
destroy()

getServletConfig()
getServletInfo()
```

Life Cycle method

Non Life Cycle method

GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

Methods of GenericServlet class

- `public void init(ServletConfig)`
- `public abstract void service(ServletRequest request, ServletResponse response)`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`
- `public ServletContext getServletContext()`
- `public String getInitParameter(String name)`
- `public Enumeration getInitParameterNames()`
- `public String getServletName()`
- `public void log(String msg)`
- `public void log(String msg, Throwable t)`

HttpServlet class

1. [HttpServlet class](#)
2. [Methods of 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.

Servlet Tests

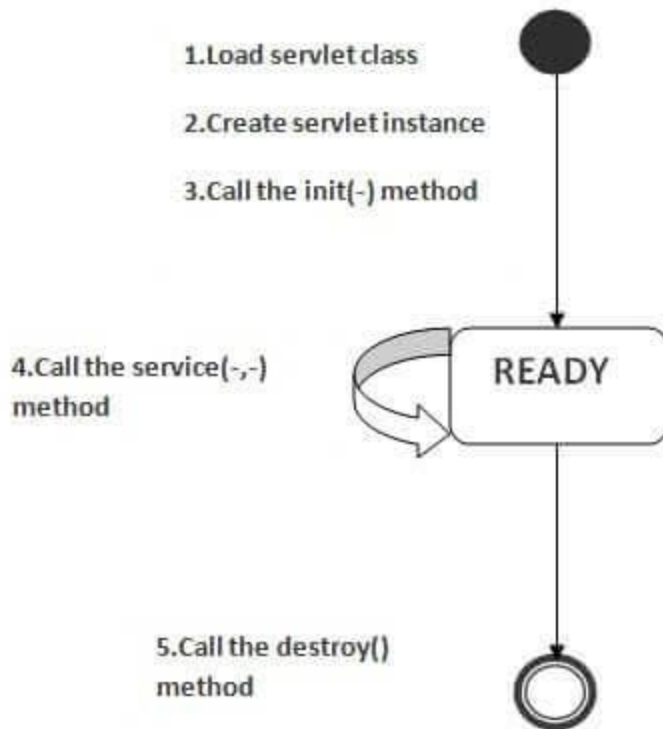
Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a Servlet

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

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

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



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

1) Servlet class is loaded

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

2) Servlet instance is created

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

3) init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method initializes 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) **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()

Creating First Servlet Application using Netbeans 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 Integrated Development Environment(IDE) is the easiest way to create Servlet Applications.

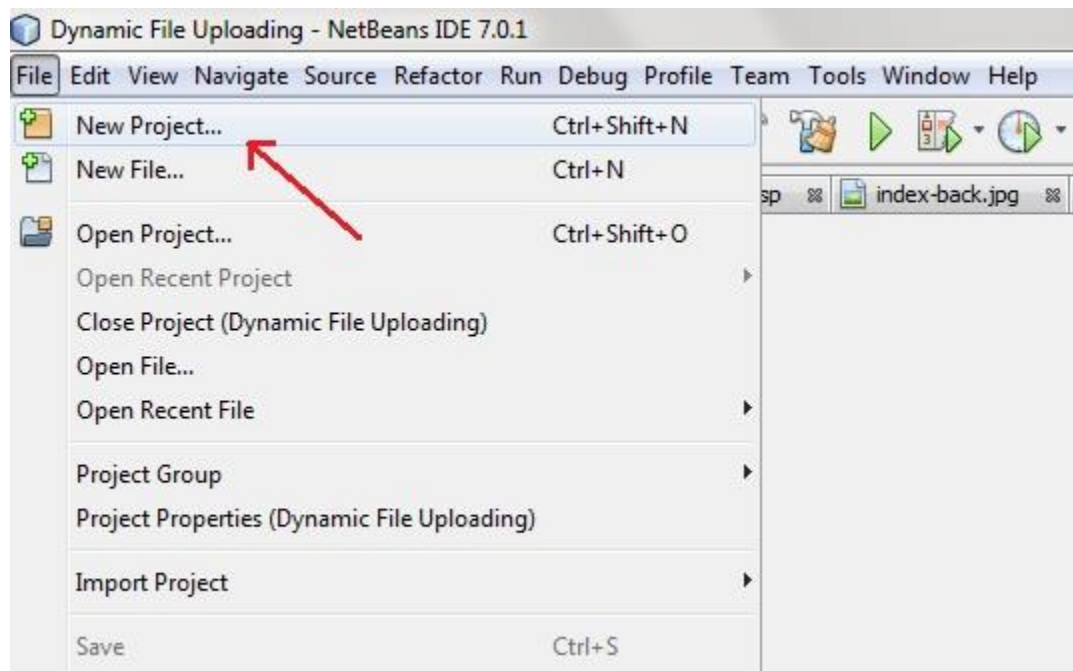
An IDE is a software application that provides facilities to computer programmers for software development.

Eclipse, MyEclipse, Netbeans are example of some popular Java IDE.

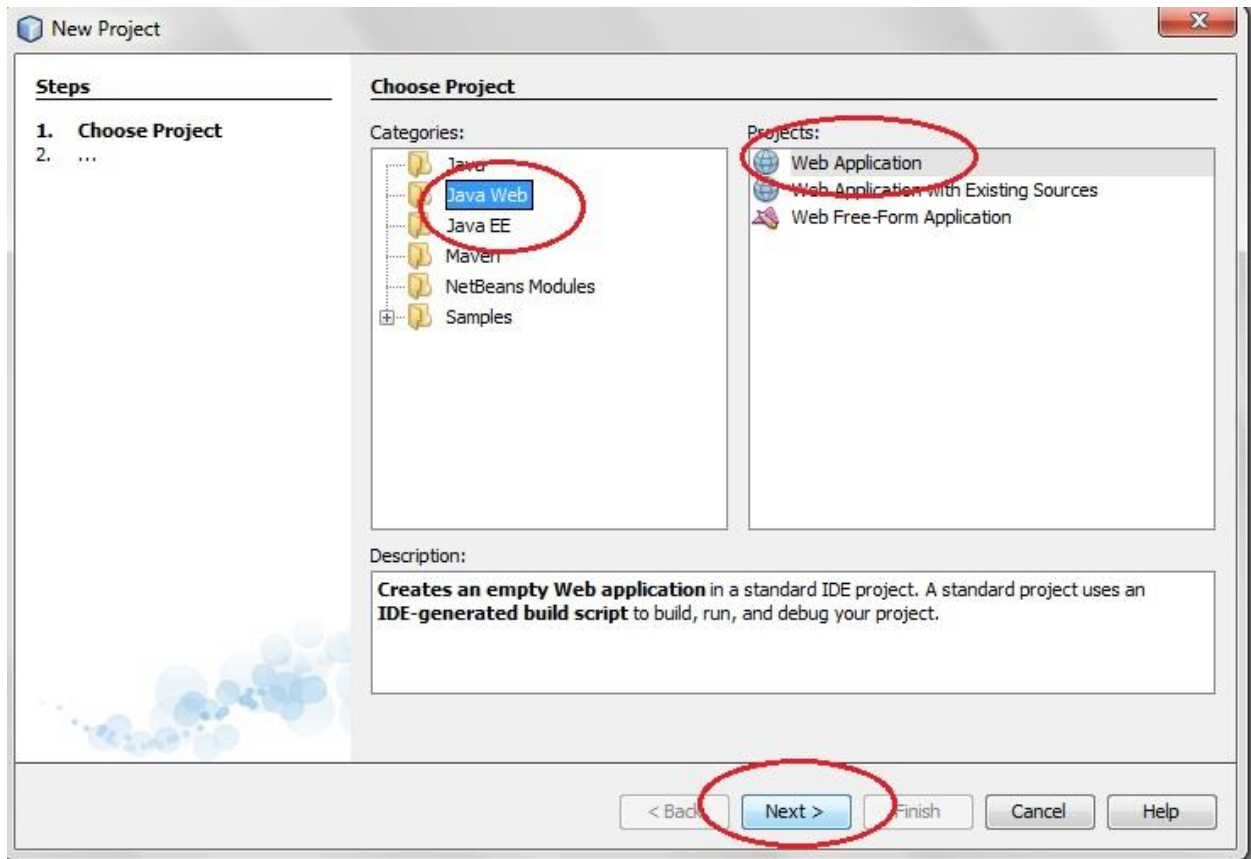
Steps to Create Servlet Application in Netbeans IDE

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

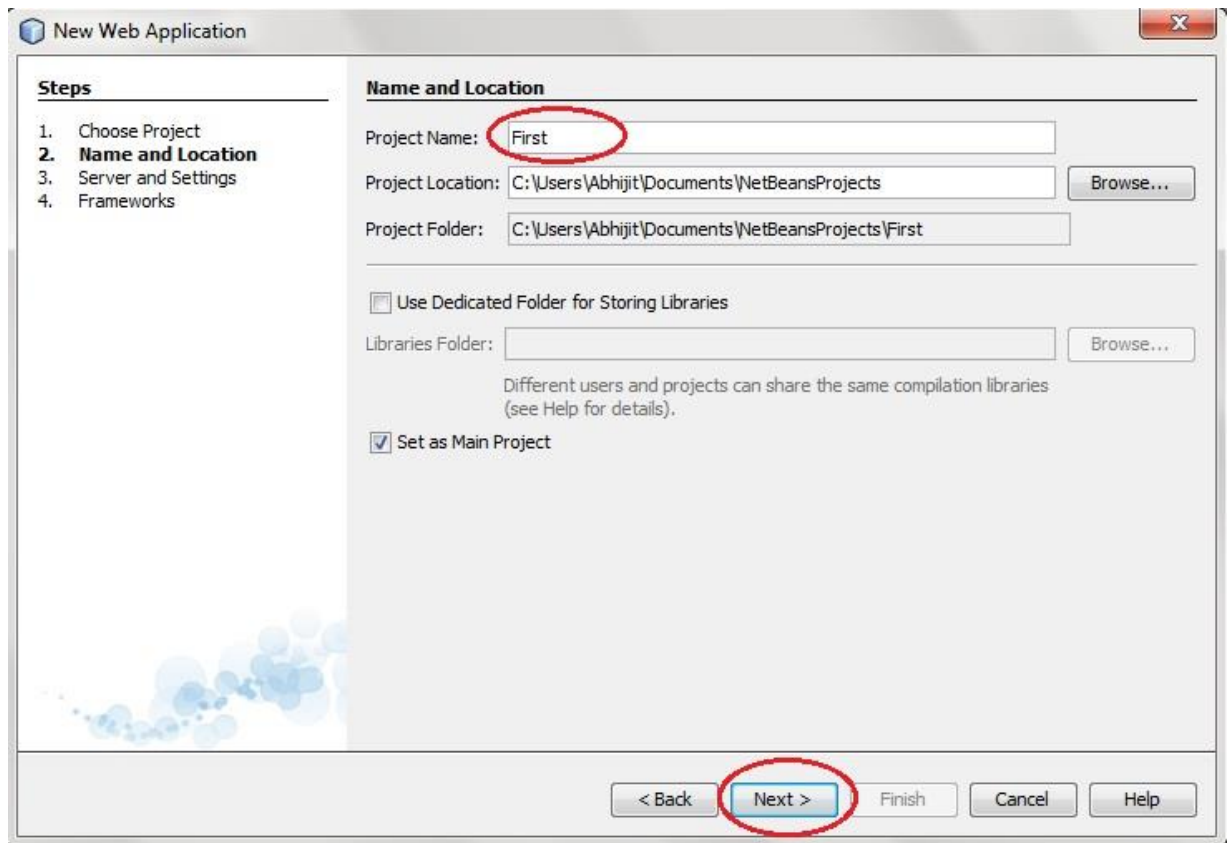
1. Open Netbeans IDE, Select **File -> New Project**



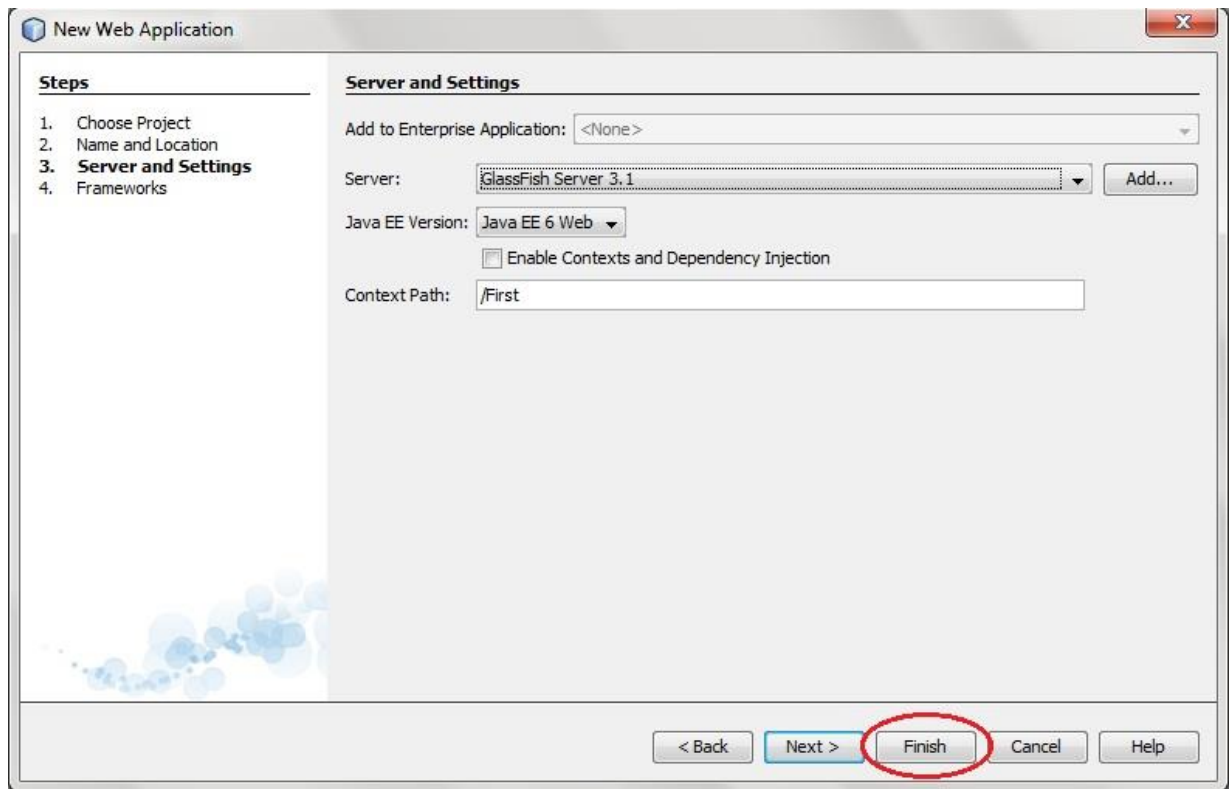
2. Select **Java Web** -> **Web Application**, then click on Next,



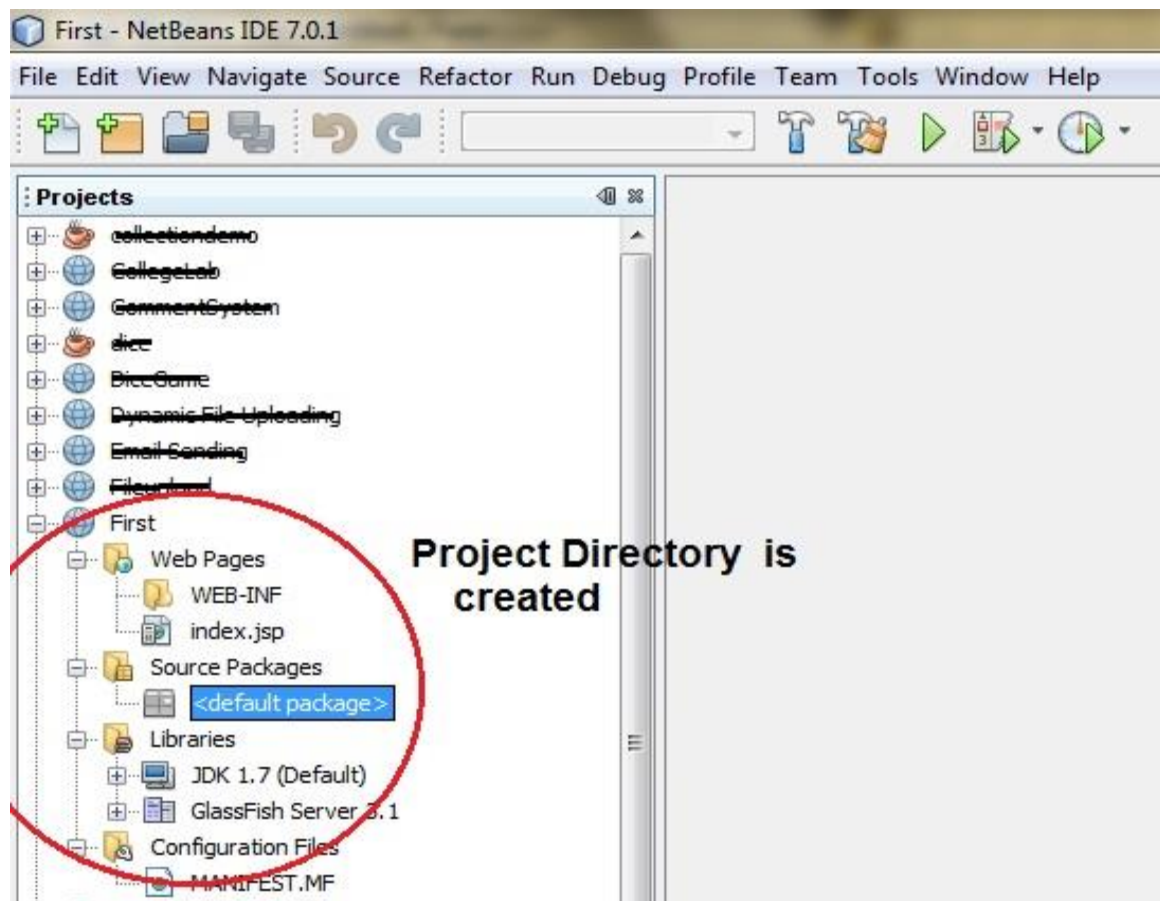
3. Give a name to your project and click on Next,



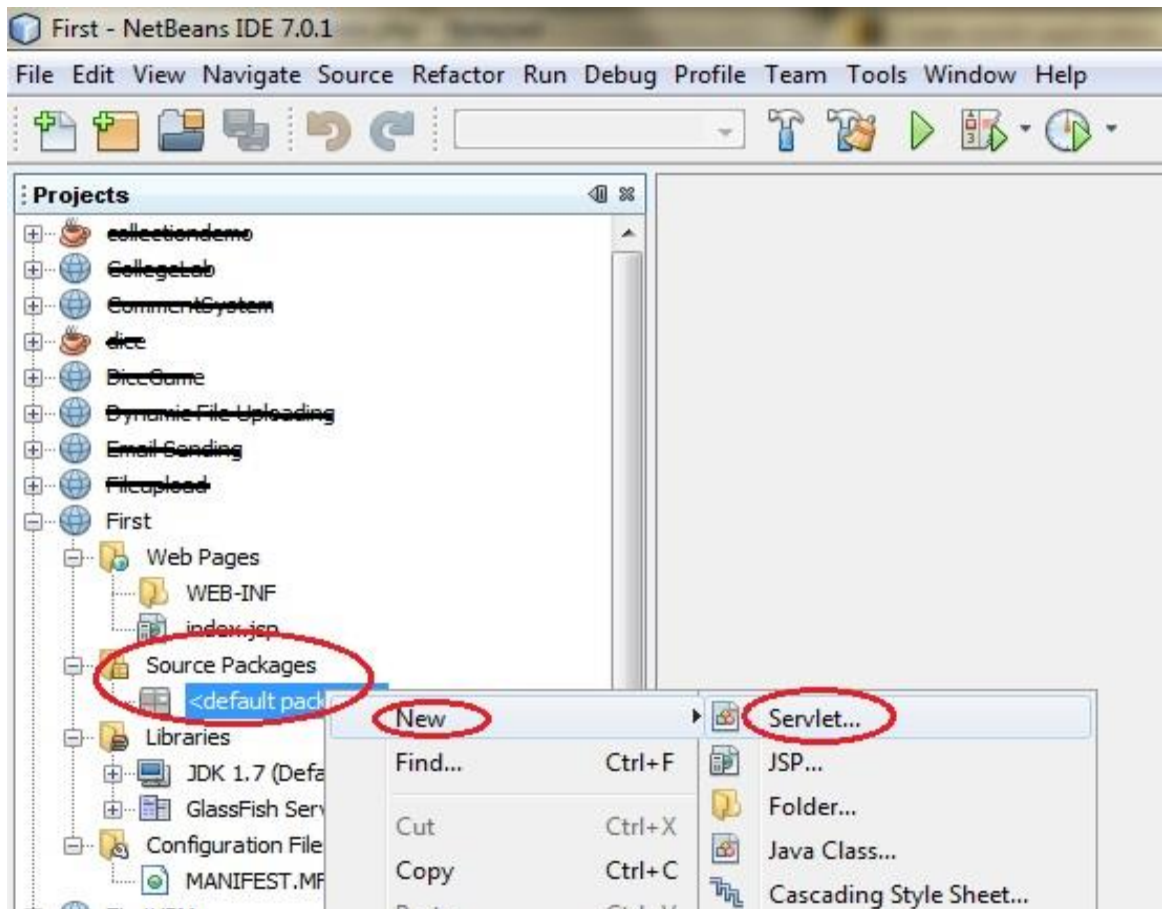
4. and then, Click **Finish**



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



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



7. Give a Name to your Servlet class file,

New Servlet

Steps

1. Choose File Type

2. **Name and Location**

3. Configure Servlet Deployment

Name and Location

Class Name: MyServlet

Project: First

Location: Source Packages

Package:

Created File: C:\Users\Abhijit\Documents\NetBeansProjects\First\src\java\MySer

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

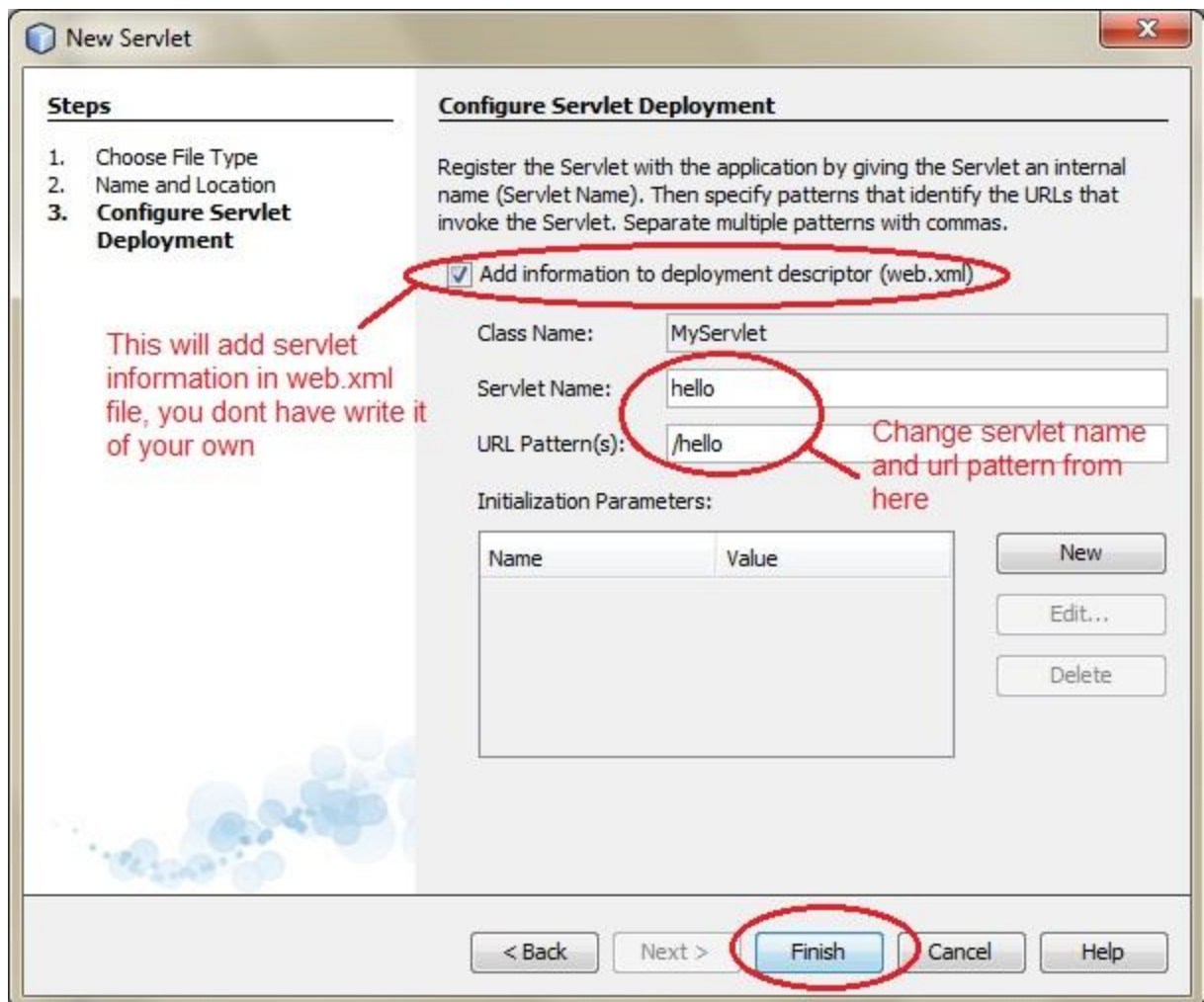
< Back

Next >

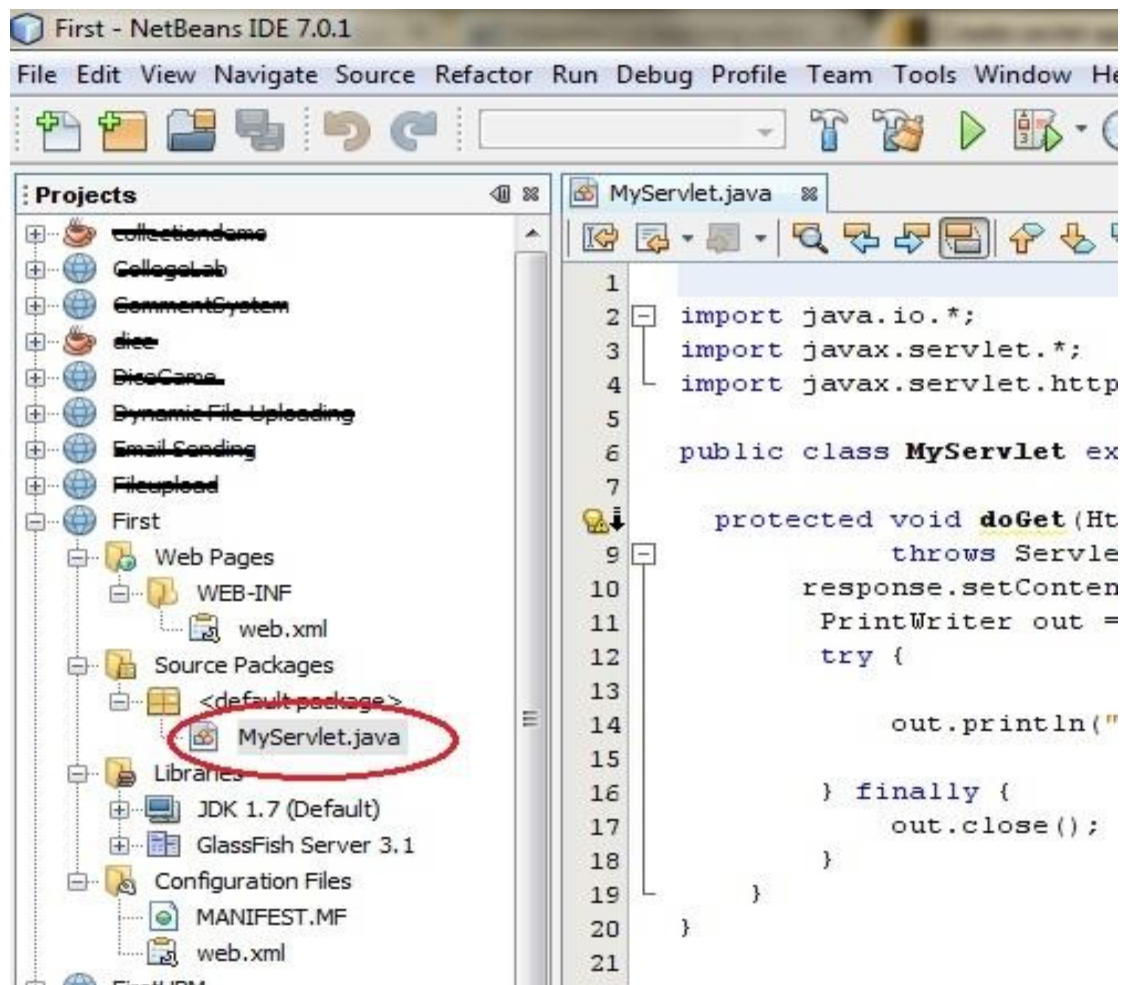
Finish

Cancel

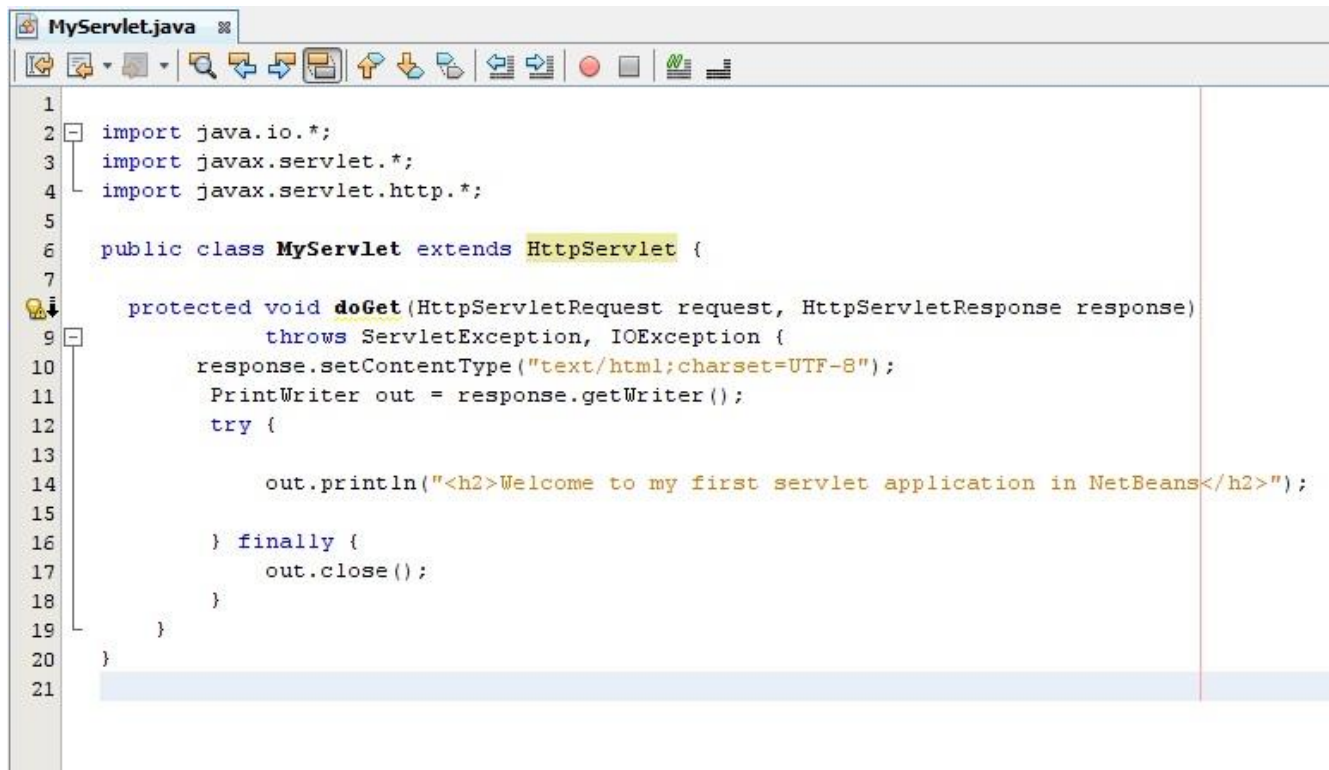
Help



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

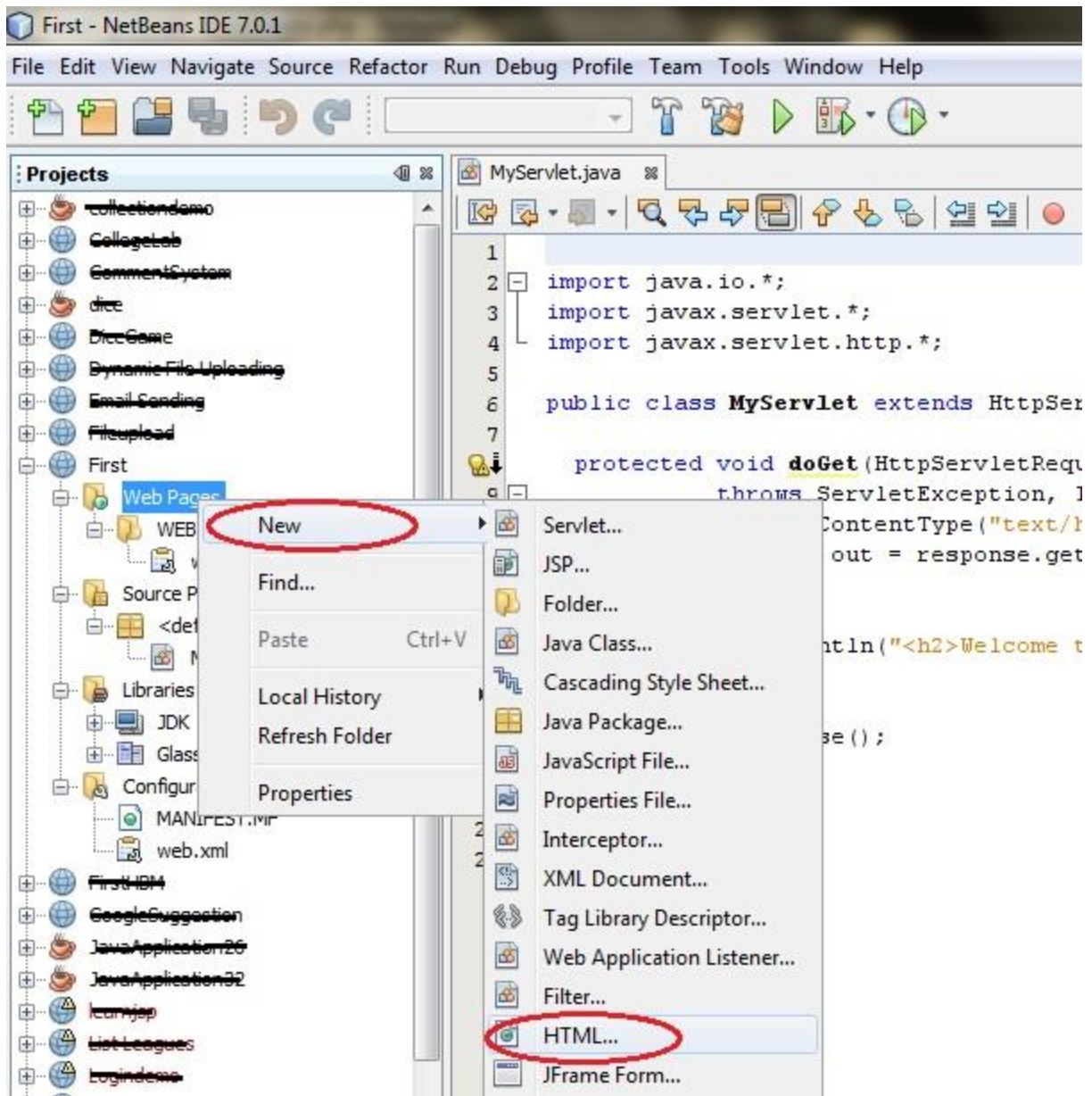


9. Write some code inside your Servlet class.

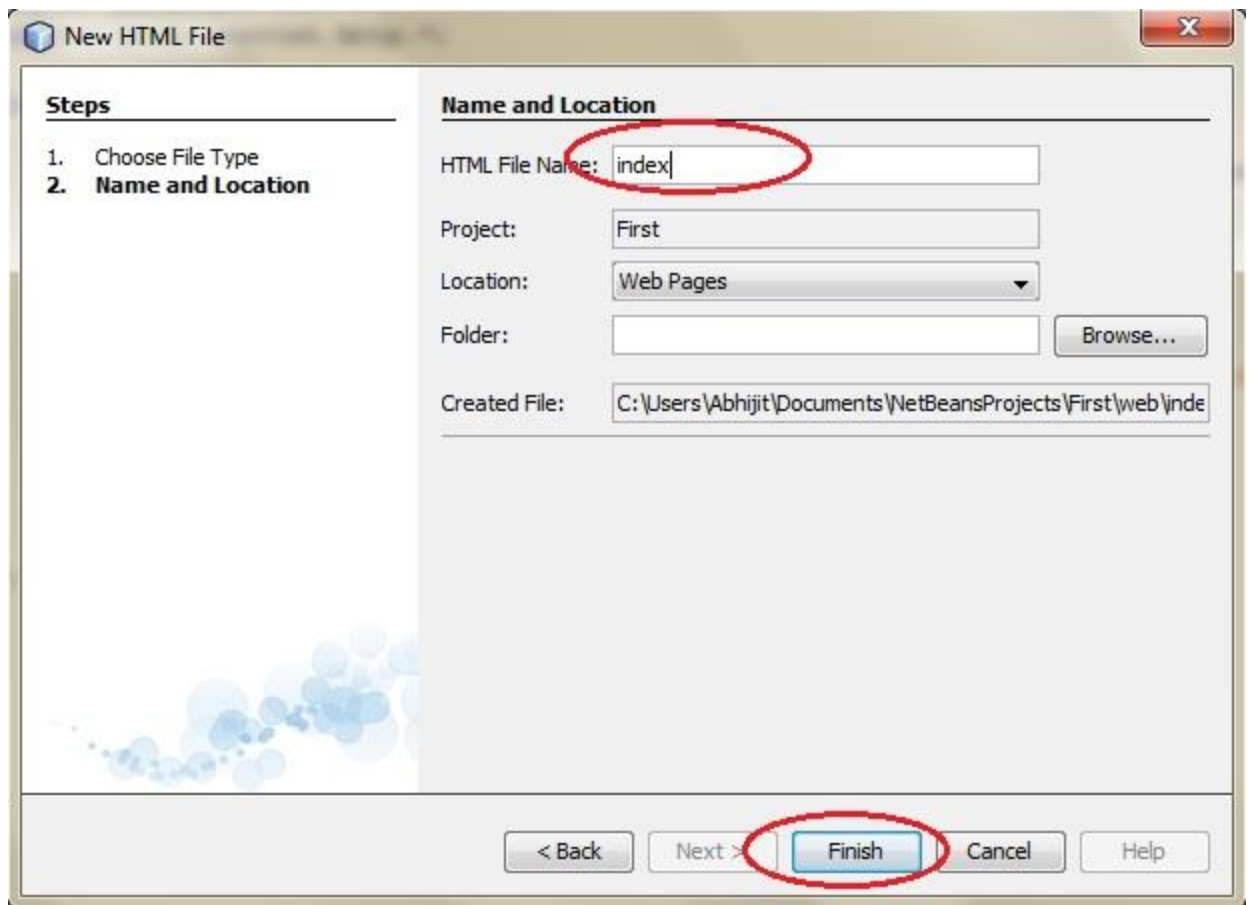


```
1
2 import java.io.*;
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5
6 public class MyServlet extends HttpServlet {
7
8     protected void doGet(HttpServletRequest request, HttpServletResponse response)
9         throws ServletException, IOException {
10         response.setContentType("text/html; charset=UTF-8");
11         PrintWriter out = response.getWriter();
12         try {
13
14             out.println("<h2>Welcome to my first servlet application in NetBeans</h2>");
15
16         } finally {
17             out.close();
18         }
19     }
20 }
21
```

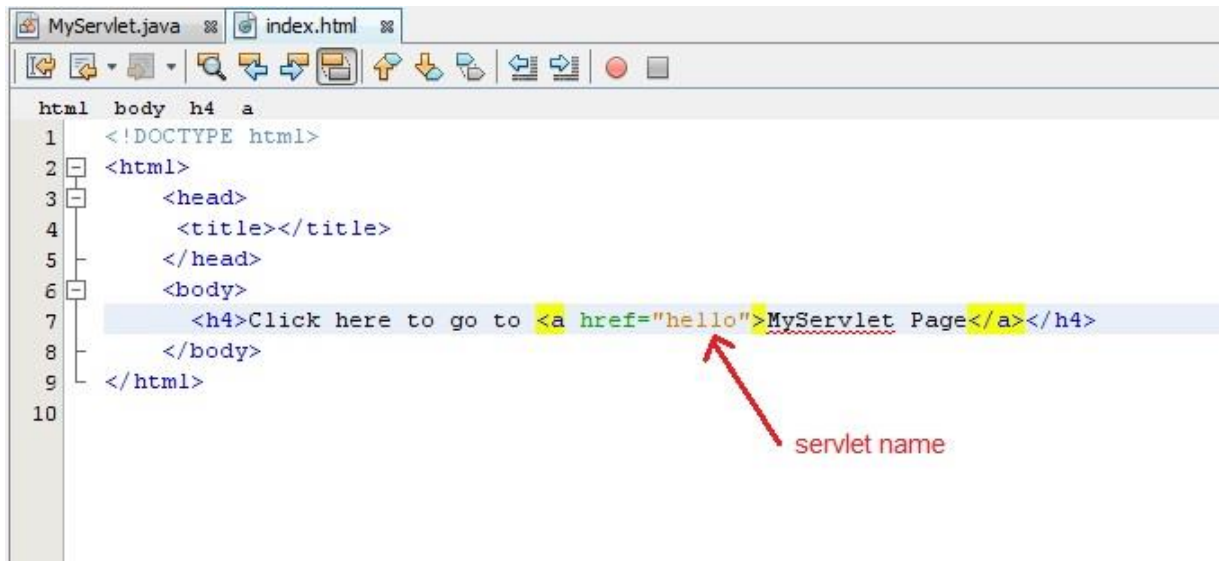
10. Create an HTML file, right click on **Web Pages** -> **New** -> **HTML**



11. 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.



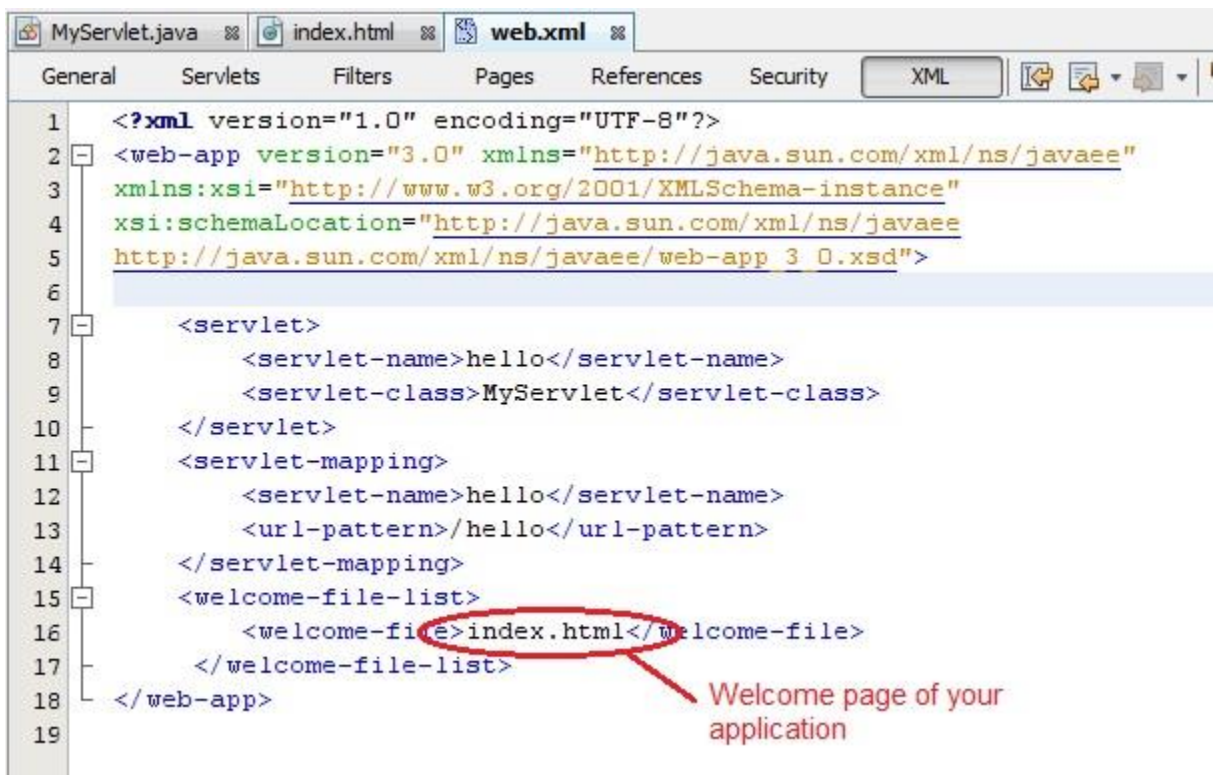
12. Write some code inside your HTML file. We have created a hyperlink to our Servlet in our HTML file.



```
html body h4 a
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5   </head>
6   <body>
7     <h4>Click here to go to <a href="hello">MyServlet Page</a></h4>
8   </body>
9 </html>
10
```

servlet name

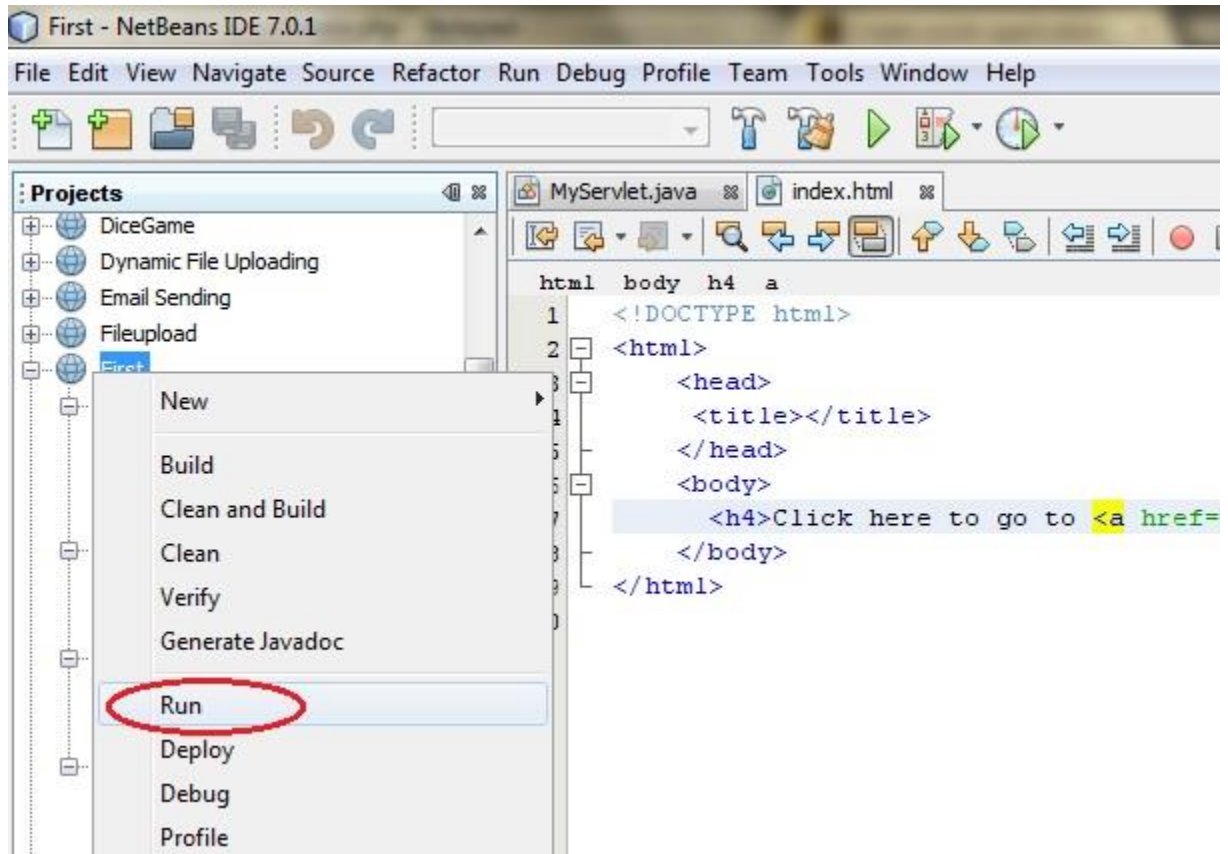
13. Edit **web.xml** file. In the web.xml file you can see, we have specified the **url-pattern** and the **servlet-name**, this means when **hello** url is accessed our Servlet file will be executed.



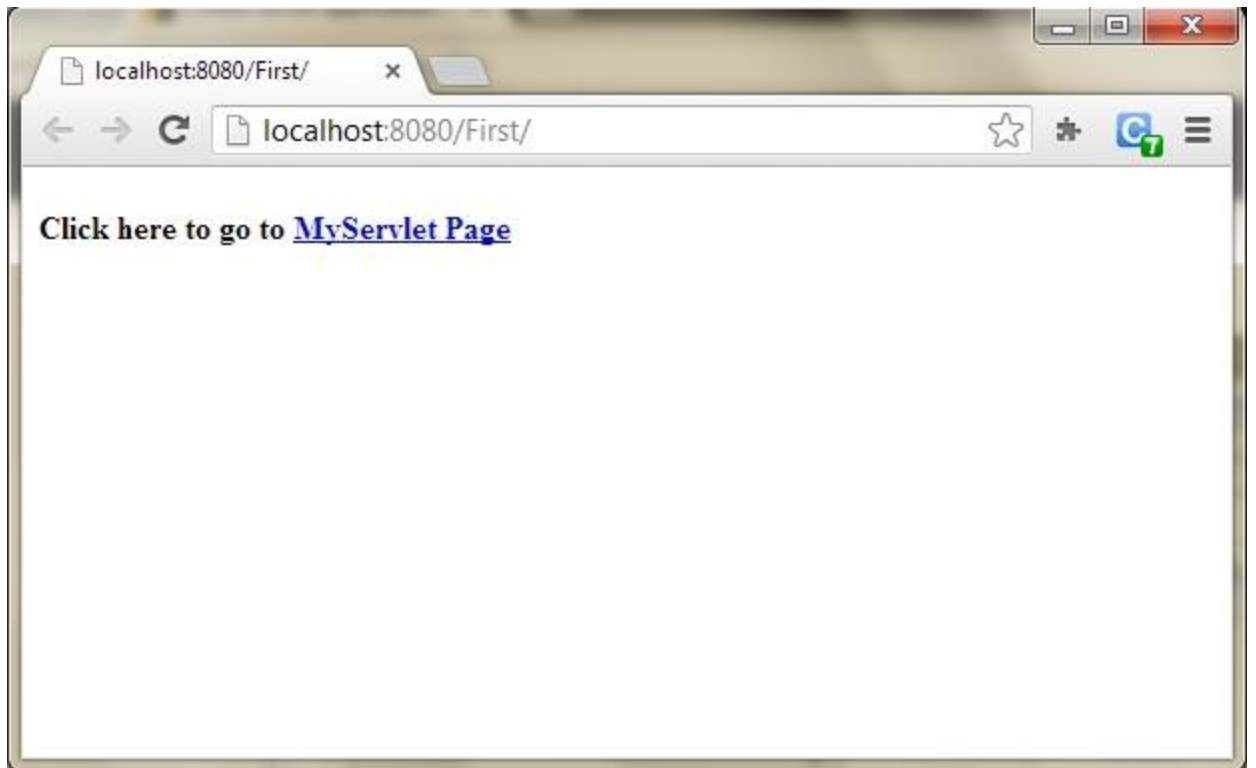
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5     http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
6
7   <servlet>
8     <servlet-name>hello</servlet-name>
9     <servlet-class>MyServlet</servlet-class>
10  </servlet>
11  <servlet-mapping>
12    <servlet-name>hello</servlet-name>
13    <url-pattern>/hello</url-pattern>
14  </servlet-mapping>
15  <welcome-file-list>
16    <welcome-file>index.html</welcome-file>
17  </welcome-file-list>
18 </web-app>
19
```

Welcome page of your application

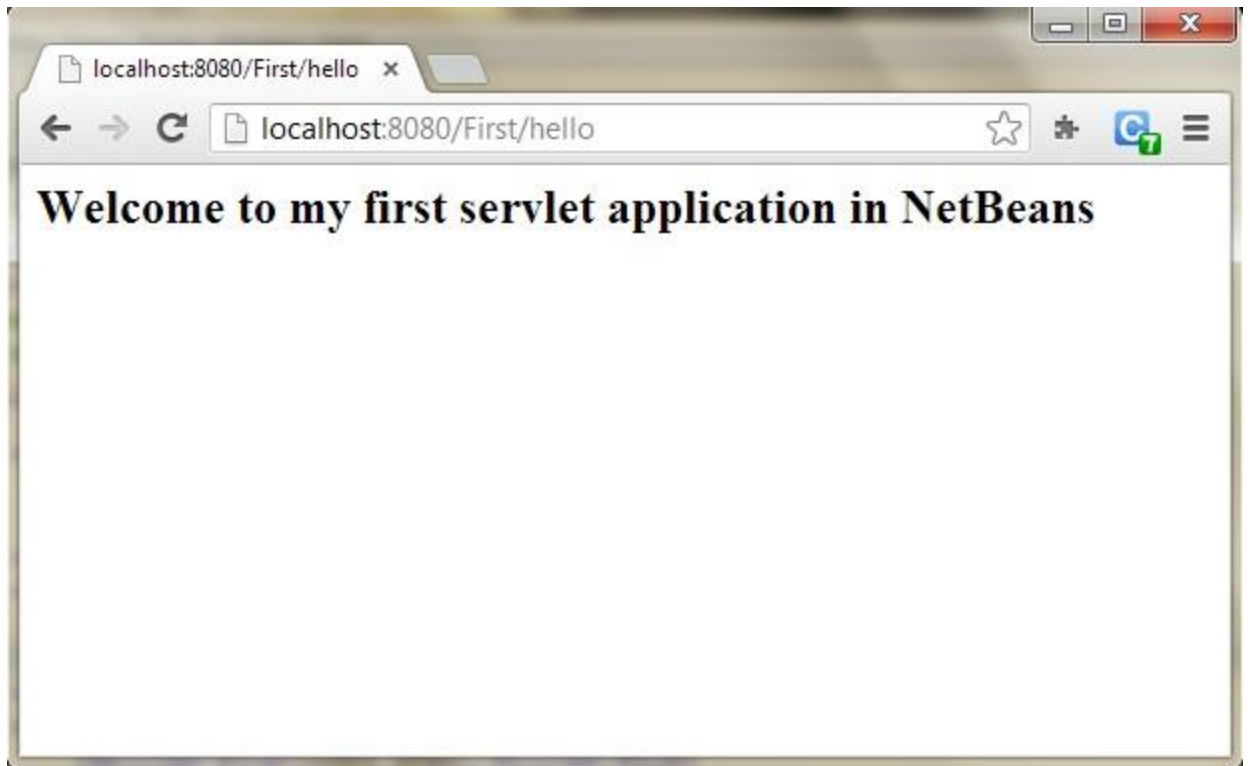
14. Run your application, right click on your Project and select **Run**



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



16. Hurray! Our First Servlet class is running.



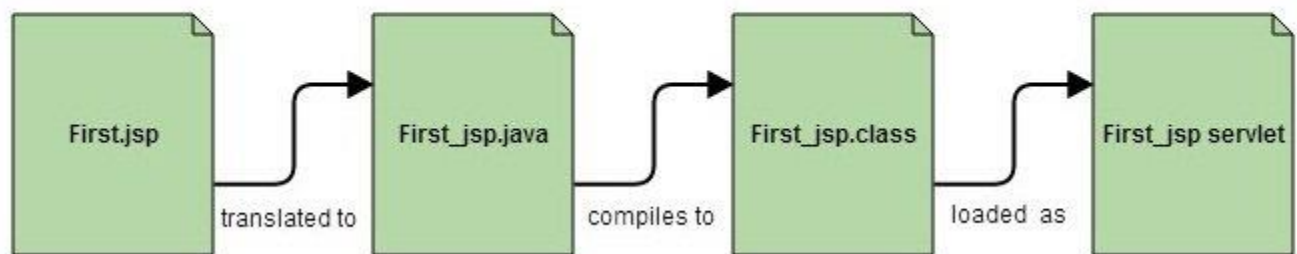
Introduction to JSP

JSP technology is used to create dynamic web applications. JSP pages are easier to maintain than a Servlet. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

In the end a JSP becomes a Servlet

JSP pages are converted into **Servlet** by the Web Container. The Container translates a JSP page into servlet **class source(.java)** file and then compiles into a Java Servlet class.



Why JSP is preferred over servlets?

- JSP provides an easier way to code dynamic web pages.
 - JSP does not require additional files like, java class files, web.xml etc
 - Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
 - JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.
-

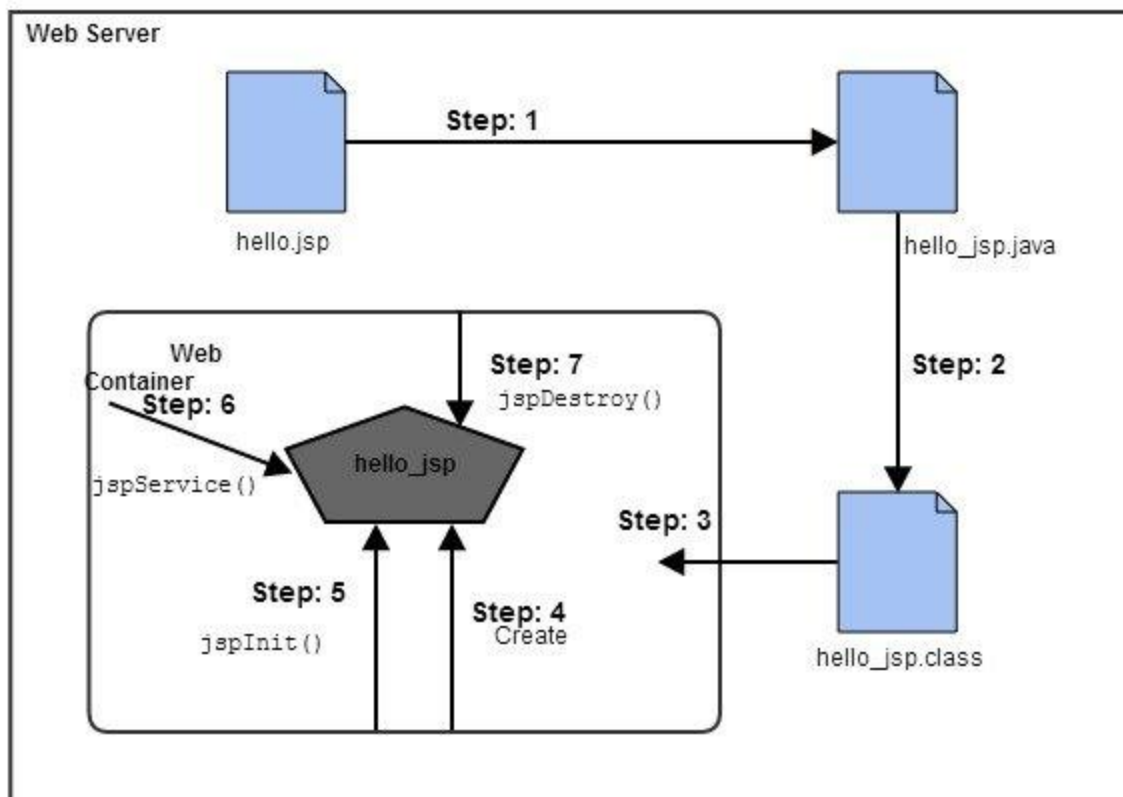
Advantage of JSP

- Easy to maintain and code.
 - High Performance and Scalability.
 - JSP is built on Java technology, so it is platform independent.
-

Lifecycle of JSP

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code. Following are the JSP Lifecycle steps:

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling `jspInit()` method
6. Request Processing by calling `_jspService()` method
7. Destroying by calling `jspDestroy()` method



Web Container translates JSP code into a **servlet class source(.java) file**, then compiles that into a java servlet class. In the third step, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.

The initialized servlet can now service request. For each request the **Web Container** call the **_jspService()** method. When the Container removes the servlet instance from service, it calls the **jspDestroy()** method to perform any required clean up.

What happens to a JSP when it is translated into Servlet

Let's see what really happens to JSP code when it is translated into Servlet. The code written inside `<% %>` is JSP code.

```
<html>

  <head>

    <title>My First JSP Page</title>

  </head>

  <%

    int count = 0;

  %>

  <body>

    Page Count is:

    <% out.println(++count); %>

  </body>

</html>
```

Copy

The above JSP page(hello.jsp) becomes this Servlet,

```
public class hello_jsp extends HttpServlet

{

    public void _jspService(HttpServletRequest
request, HttpServletResponse response)
```

```

throws
IOException, ServletException

{

    PrintWriter out = response.getWriter();

    response.setContentType("text/html");

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

    int count=0;

    out.write("Page count is:");

    out.print(++count);

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

}
}

```

This is just to explain, what happens internally. As a JSP developer, you do not have to worry about how a JSP page is converted to a Servlet, as it is done automatically by the web container.

Creating a JSP Page

A JSP page looks similar to an [HTML page](#), but a JSP page also has Java code in it. We can put any regular Java Code in a JSP file using

a **scriptlet tag** which start with `<%` and ends with `%>`. JSP pages are used to develop dynamic responses.

To learn HTML, go to [HTML Interactive Course](#) and learn HTML while practicing it side by side.

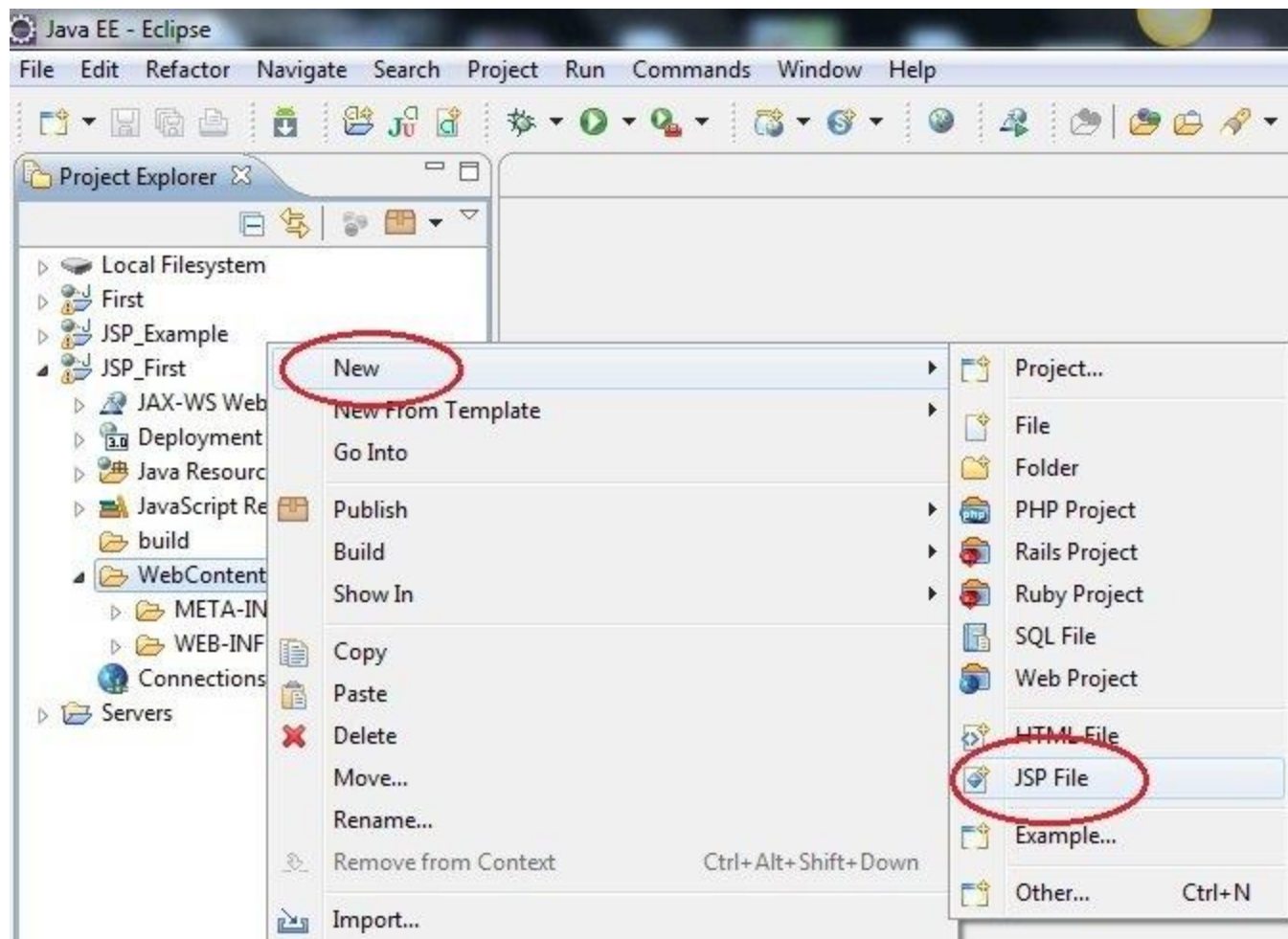
Example of creating a JSP Page in Eclipse

- Open Eclipse, Click on **New** → **Dynamic Web Project**
-

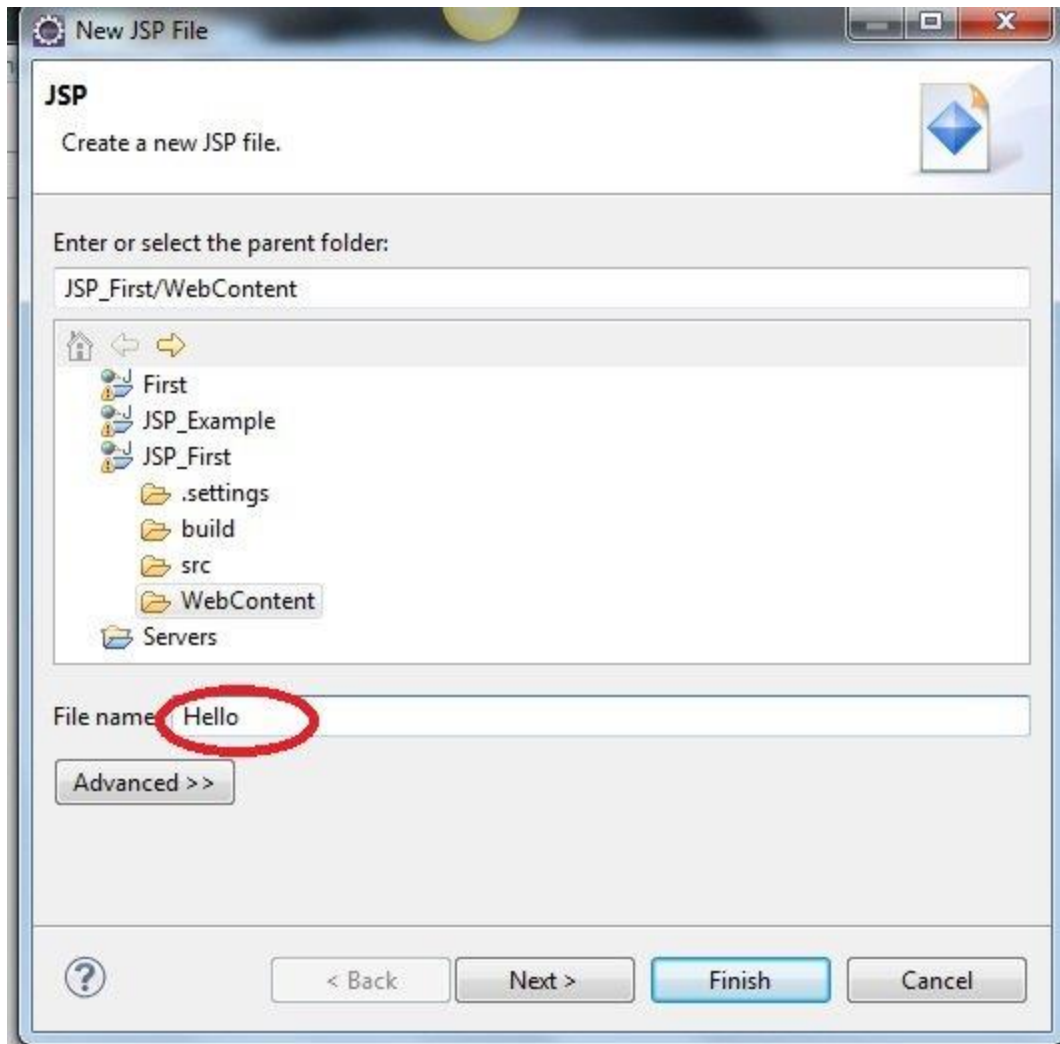
- Give a name to your project and click on OK
-

- You will see a new project created in Project Explorer
-

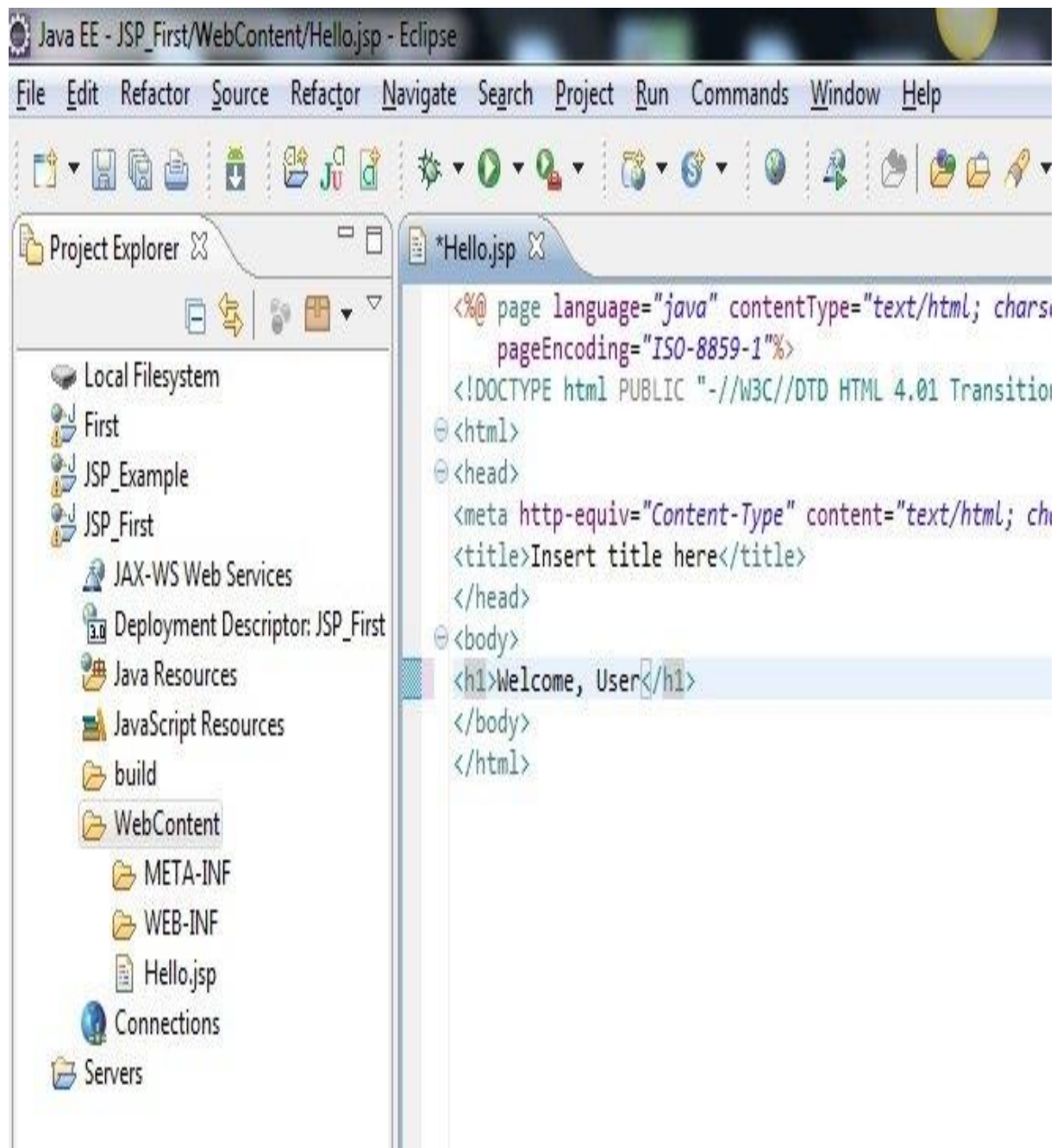
- To create a new JSP file right click on Web Content directory, **New** → **JSP file**



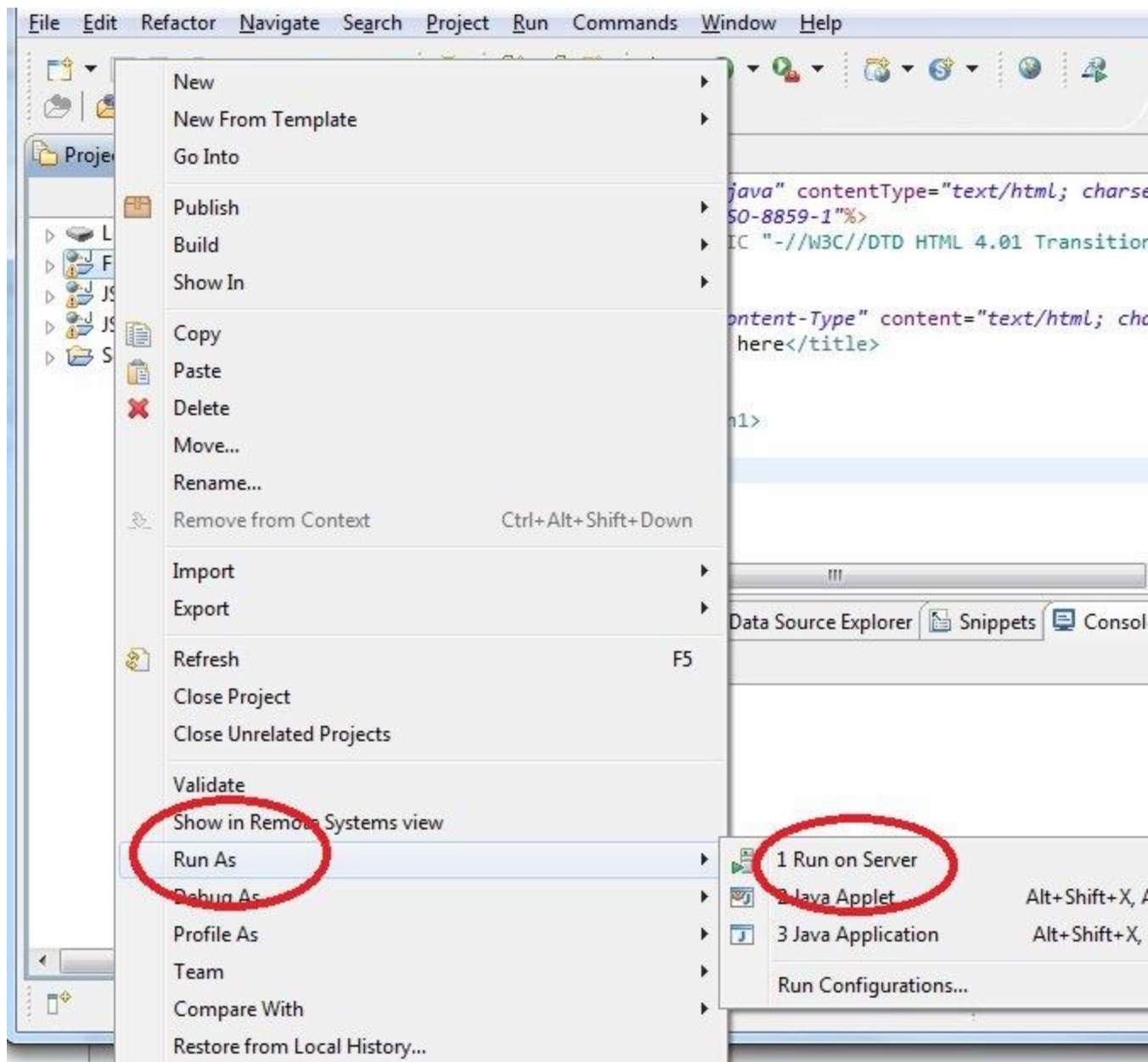
- Give a name to your JSP file and click Finish.



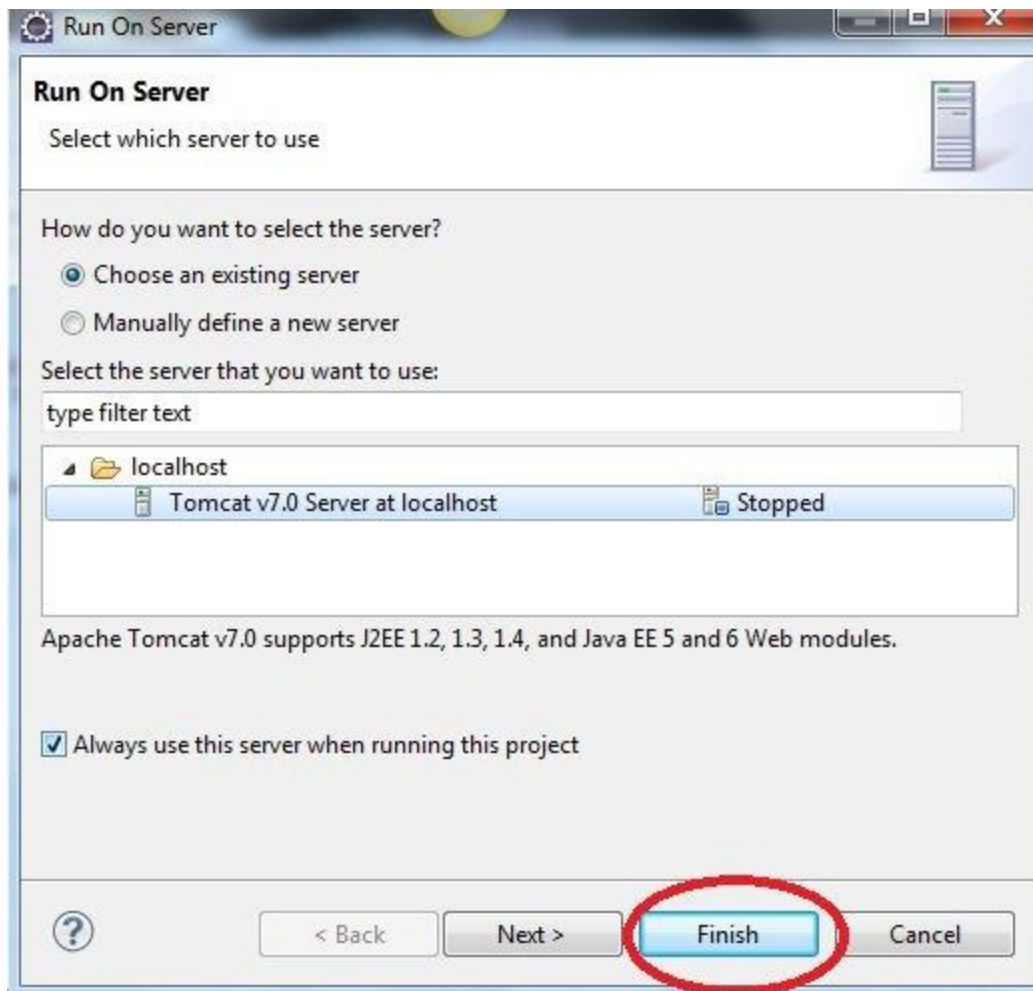
- Write something in your JSP file. The complete HTML and the JSP code, goes inside the `<body>` tag, just like HTML pages.



- To run your project, right click on **Project**, select **Run As** → **Run on Server**



- To start the server, Choose existing server name and click on finish



- See the Output in your browser.



JSP Include Directive

The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:

```
<%@ include file="filename.jsp" %>
```

Copy

Example of include directive

welcome.jsp

```
<html>

  <head>

    <title>Welcome Page</title>

  </head>


  <body>

    <%@ include file="header.jsp" %>

    Welcome, User

  </body>

</html>
```

Copy

header.jsp

```
<html>

  <body>

    
```

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

Copy

The example above is showcasing a very standard practice. Whenever we are building a web application, with webpages, all of which have the top navbar and bottom footer same. We make them as separate jsp files and include them using the `include` directive in all the pages. Hence whenever we have to update something in the top navbar or footer, we just have to do it at one place. Handy, isn't it?

One more standard application of `include` directive is, if you create a separate jsp file, with some commonly used functions, kind of like a util jsp file. Which can be included in the web pages wherever you want to use those functions.



Similarly, there are many ways in which this directive proves to be quite useful in giving a structure to your web application code.

JSP Taglib Directive

The taglib directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library. JavaServer Pages Standard Tag Library (JSTL), is a collection of useful JSP tags, which provides many commonly used core functionalities. It has support for many general, structural tasks such as iteration and conditionals, readymade tags for manipulating XML documents, internationalization tags, and for performing SQL operations. Syntax of taglib directive is:

```
<%@ taglib prefix="prefixOfTag"
uri="uriOfTagLibrary" %>
```

Copy

The prefix is used to distinguish the custom tag from other library custom tag. Prefix is prepended to the custom tag name. Every custom tag must have a prefix.

The URI is the unique name for Tag Library.

You can name the prefix anything, but it should be unique.

JSP: Using Taglib Directive

To use the JSTL in your application you must have the [jstl.jar](#) in your webapps [/WEB-INF/lib](#) directory. Download the jar file from [Apache Standard Taglib](#) page.

There are many readymade JST Libraries available which you use to make your life easier. Following is a broad division on different groups of JST libraries :

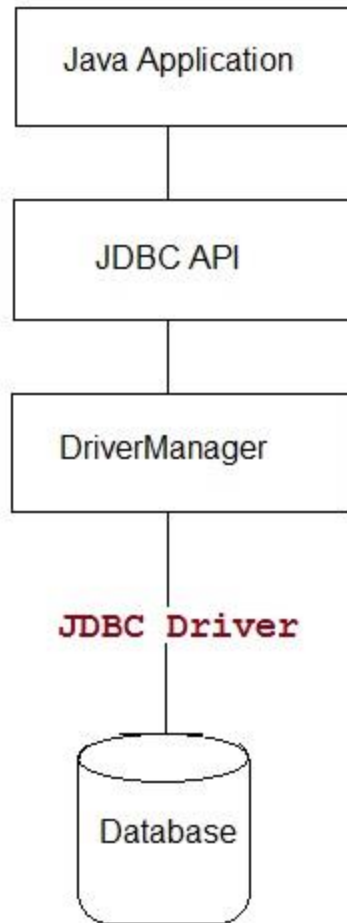
1. Core Tags - URI → <http://java.sun.com/jsp/jstl/core>
2. Formatting Tags - URI → <http://java.sun.com/jsp/jstl/fmt>
3. SQL Tags - URI → <http://java.sun.com/jsp/jstl/sql>
4. XML Tags - URI → <http://java.sun.com/jsp/jstl/xml>
5. JSTL Functions - URI → <http://java.sun.com/jsp/jstl/functions>

Introduction to JDBC

Java Database Connectivity(JDBC) is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

The JDBC API consists of classes and methods that are used to perform various operations like: connect, read, write and store data in the database. In this tutorial we will learn the JDBC with examples.

You can get idea of how JDBC connect Java Application to the database by following image.



What's new in JDBC 4.0

Java introduced **JDBC 4.0**, a new version which is advance specification of JDBC. It provides the following advance features

- Connection Management
- Auto loading of Driver Interface.
- Better exception handling
- Support for large object
- Annotation in SQL query.

JDBC Driver

JDBC Driver is required to establish connection between application and database. It also helps to process SQL requests and generating result. The following are the different types of driver available in JDBC which are used by the application based on the scenario and type of application.

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

DriverManager class

In Java, the DriverManager class it an interface between the User and the Driver. This class is used to have a watch on driver which is been used for establishing the connection between a database and a driver. The DriverManager class have a list of Driver class which are registered and are called as `DriverManager.registerDriver()`.

S.No.	Method	Description
1	public static void registerDriver(Driver driver)	It is used for Registering the Driver with the Driver Manager.
2	public static void deregisterDriver(Driver driver)	It is used for Deregistering the Driver with the Driver Manager.
3	public static Connection getConnection(String Url)	It is used for establishing a connection with the given URL.
4	public static Connection getConnection(String Url, String username, String password)	

JDBC API

JDBC API is mainly divided into two package. Each when we are using JDBC, we have to import these packages to use classes and interfaces in our application.

1. `java.sql`
2. `javax.sql`

java.sql package

This package include classes and interface to perform almost all JDBC operation such as creating and executing SQL Queries.

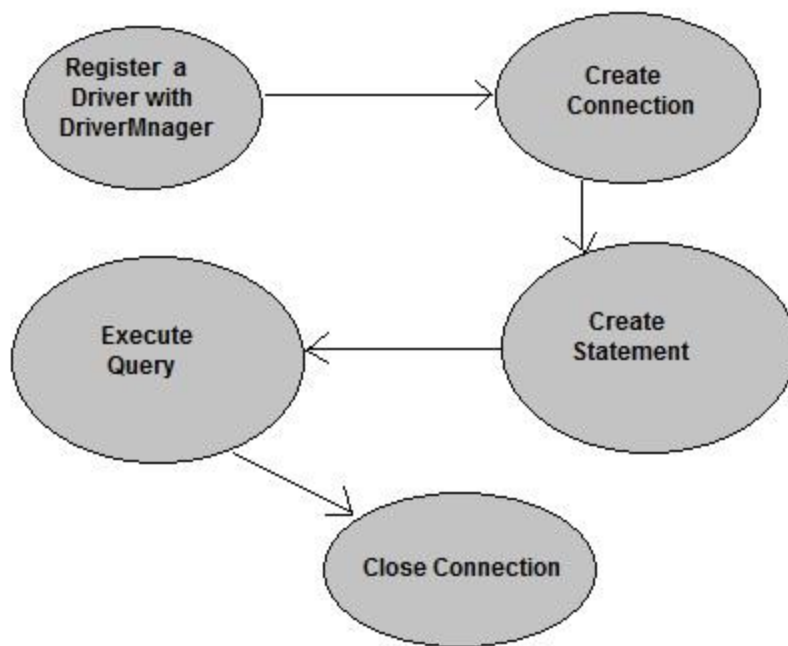
The javax.sql package

This package is also known as JDBC extension API. It provides classes and interface to access server-side data.

Steps to connect a Java Application to Database

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection



Managing Session in Servlets

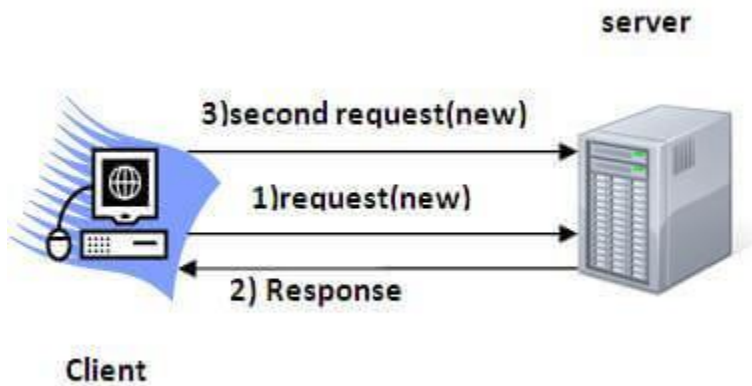
We all know that **HTTP** is a stateless protocol. All requests and responses are independent. But sometimes you need to keep track of client's activity across multiple requests. For eg. When a User logs into your website, not matter on which web page he visits after logging in, his credentials will be with the server, until he logs out. So this is managed by creating a session.

Session Management is a mechanism used by the **Web container** to store session information for a particular user. There are four different techniques used by Servlet application for session management. They are as follows:

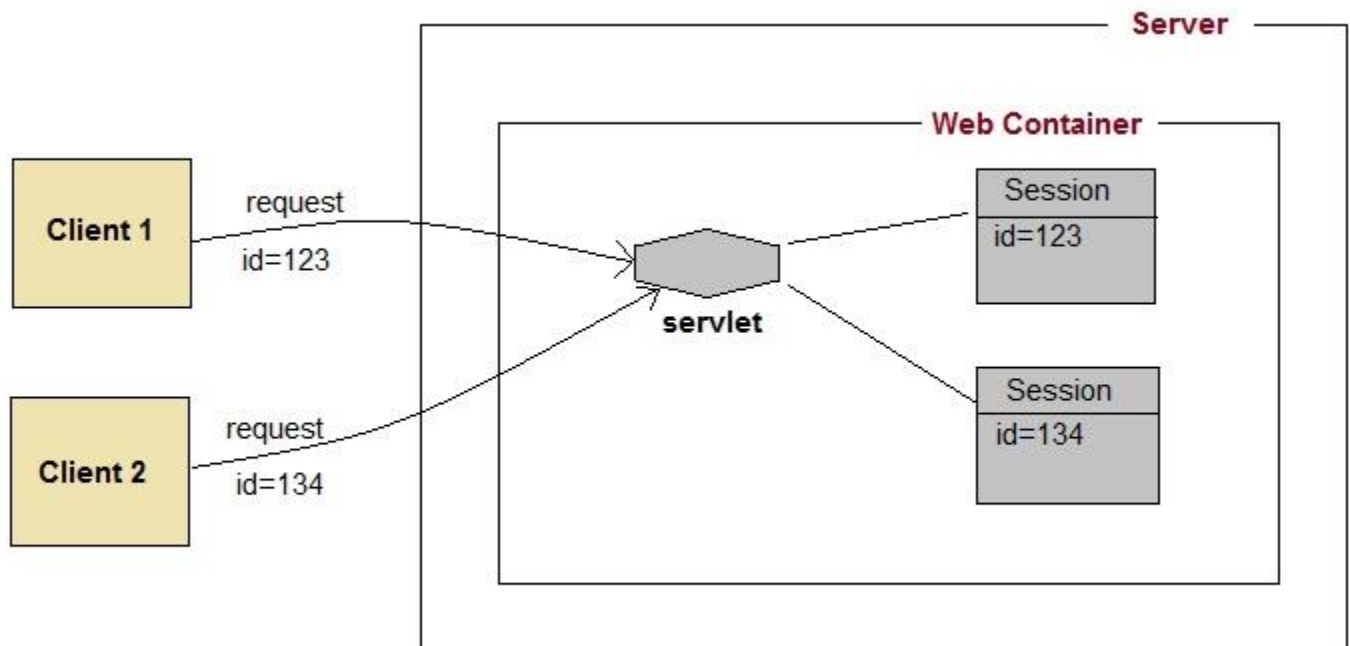
1. **Cookies**
2. **Hidden form field**
3. **URL Rewriting**
4. **HttpSession**

Session is used to store everything that we can get from the client from all the requests the client makes.

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



How Session Works



The basic concept behind session is, whenever a user starts using our application, we can save a unique identification information about him, in an object which is available

throughout the application, until its destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit from your application, destroy the object with his information.

Using Cookies for Session Management in Servlet

Cookies are small pieces of information that are sent in response from the web server to the client. **Cookies** are the simplest technique used for storing client state.

Cookies are stored on client's computer. They have a lifespan and are destroyed by the client browser at the end of that lifespan.

Using Cookies for storing client state has one shortcoming though, if the client has turned off COokie saving settings in his browser then, client state can never be saved because the browser will not allow the application to store cookies.

Servlet: Cookies API

Cookies are created using **Cookie** class present in Servlet API. Cookies are added to **response** object using the `addCookie()` method. This method sends cookie information over the HTTP response stream. `getCookies()` method is used to access the cookies that are added to response object.

Creating a new Cookie

```
Cookie ck = new Cookie("username", name);
```

creating a new cookie object

Setting up lifespan for a cookie

```
ck.setMaxAge(30*60);
```

setting maximum age of cookie

Sending the cookie to the client

```
response.addCookie(ck);
```

adding cookie to response object

Getting cookies from client request

```
Cookie[] cks = request.getCookies();
```

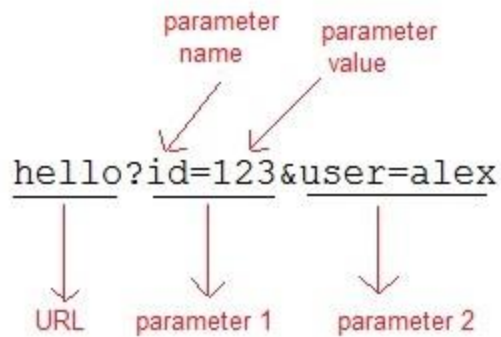
getting the cookie for request object

Using URL Rewriting for Session Management in Servlet

If the client has disabled cookies in the browser then session management using cookie wont work. In that case **URL Rewriting** can be used as a backup. **URL rewriting** will always work.

In URL rewriting, a token(parameter) is added at the end of the URL. The token consist of name/value pair seperated by an `equal(=)` sign.

For Example:



When the User clicks on the URL having parameters, the request goes to the **Web Container** with extra bit of information at the end of URL. The **Web Container** will fetch the extra part of the requested URL and use it for session management.

The `getParameter()` method is used to get the parameter value at the server side.

Using Hidden Form Field for Session Management in Servlet

Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieved from another servlet.

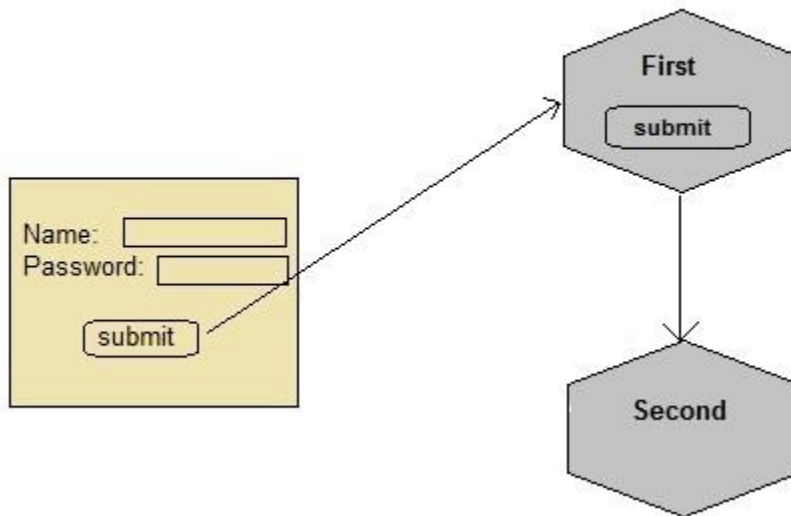
Advantages of Using Hidden Form Field for Session Management

- Does not have to depend on browser whether the cookie is disabled or not.
- Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.

Disadvantage of Using Hidden Form Field for Session Management

- Extra form submission is required on every page. This is a big overhead.

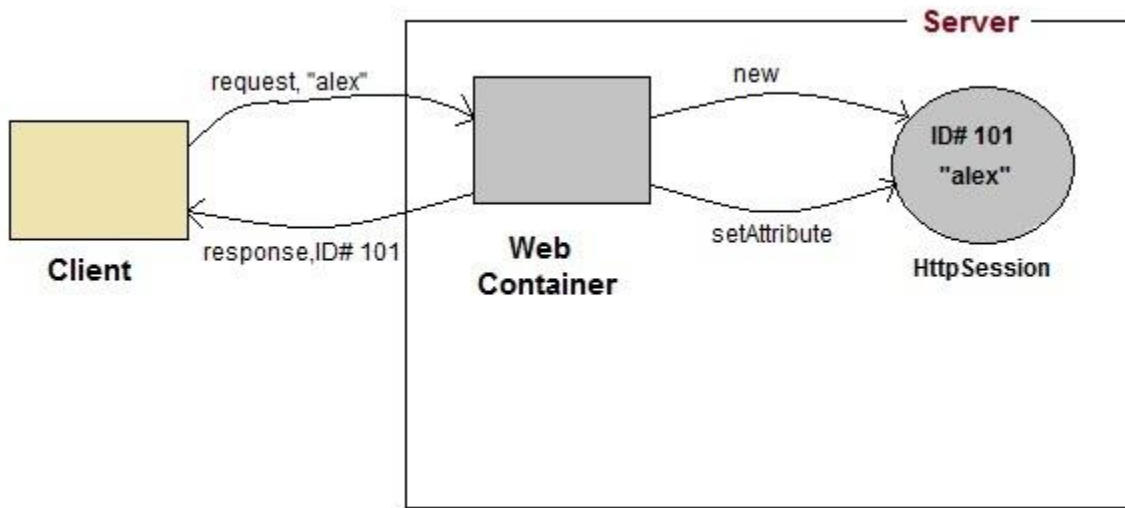
Example demonstrating usage of Hidden Form Field for Session



Servlet: What is HttpSession?

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the `getSession()` method of the **HttpServletRequest** object.

Servlet: How HttpSession works



1. On client's first request, the **Web Container** generates a unique session ID and gives it back to the client with response. This is a temporary session created by web container.
2. The client sends back the session ID with each request. Making it easier for the web container to identify where the request is coming from.
3. The **Web Container** uses this ID, finds the matching session with the ID and associates the session with the request.

Servlet: HttpSession Interface

Creating a new session

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

getSession() method returns a session. If the session already exist, it return the existing session else create a new session

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

getSession(true) always return a new session

Getting a pre-existing session

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

return a pre-existing session

Destroying a session

```
session.invalidate();
```

destroy a session

Create an Application to add and multiply two no's to the servlet to the user in apache tomcat server

Index.html

```
<!DOCTYPE html>

<html>

<head>

<title>Practical Work</title>
```

```
</head>

<body>

<form action="add_me">

<label>First number </label> <input type="text" name="num1"/>
<br/><br/>

<label>Second number </label> <input type="text"
name="num2"/> <br/><br/>

<button type="submit" name="calculate">Product and Sum
</button><br/>

</form>

</body>

</html>
```

Create Servlet Class

```
package com.programmerbay;

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

import javax.servlet.http.*;

public class Add_Numbers extends HttpServlet{

public void service(HttpServletRequest
request,HttpServletResponse response) throws IOException
{

int num1 = Integer.parseInt(request.getParameter("num1"));
```

```
int num2 = Integer.parseInt(request.getParameter("num2"));

int sum = num1 + num2;

int product = num1 * num2;

PrintWriter output = response.getWriter();

output.println("The Answer :"+sum+"\n The product
:"+product);

}

}
```

Configure the web.xml file which you find in web content folder:-

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
id="WebApp_ID" version="3.1">

<servlet>

<servlet-name>Add</servlet-name>

<servlet-class>com.programmerbay.Add_Numbers</servlet-class>

</servlet>

<servlet-mapping>
```



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