Unit – 4 Servlets and JSP [14 Hrs.]

Servlets

Servlet technology is used to create a web application. It 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.**

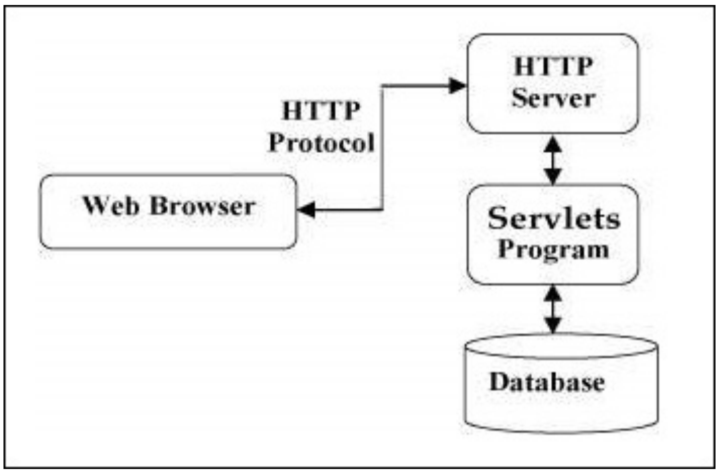
There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

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.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Limitations of CGI

Server has to create a new CGI process for every client request. For example, If 100 users are accessing the web application, then the server has to create 100 CGI processes to handle the request made by them. Since a server has limited resources, creating new process every time for a new request is not a viable option, this imposed the limitation on server, due to that the server cannot handle more than a specified number of users at the same time.

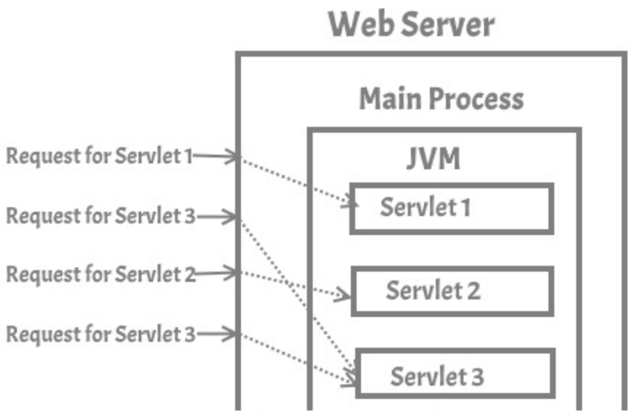
How Servlet is better than CGI

CGI programs are handled by a new process every time a new request has been made. **Unlike CGI, the servlet programs are handled by separate threads that can run concurrently more efficiently.**

CGI program can be written in any programming language that makes it mostly platform dependent as not all programming languages are platform independent. Servlet only uses Java as programming language that makes it platform independent and portable. Another benefit of using java is that the servlet can take advantage of the object oriented programming features of java.

How Servlet Works

As mentioned above that concurrent requests to the server are handled by threads, here is the graphical representation of the same –



Here dotted lines represent threads.

Features/Advantages of Servlet

1. Portable:

Servlet uses Java as a programming language, Since java is platform independent, the same holds true for servlets. For example, you can create a servlet on Windows operating system that users GlassFish as web server and later run it on any other operating system like Unix, Linux with Apache tomcat web server, this feature makes servlet portable and this is the main advantage servlet has over CGI.

2. Efficient and scalable:

Once a servlet is deployed and loaded on a web server, it can instantly start fulfilling request of clients. The web server invokes servlet using a lightweight thread so multiple client requests can be fulling by servlet at the same time using the multithreading feature of Java. Compared to CGI where the server has to initiate a new process for every client request, the servlet is truly efficient and scalable.

3. Robust:

By inheriting the top features of Java (such as Garbage collection, Exception handling, Java Security Manager etc.) the servlet is less prone to memory management issues and memory leaks. This makes development of web application in servlets secure and less error prone.

4. Better performance: because it creates a thread for each request, not process.
5. Secure: because it uses java language.

Servlet API

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

Interfaces in javax.servlet package

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

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

Classes in javax.servlet package

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

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

Interfaces in javax.servlet.http package

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

1. HttpServletRequest

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

Classes in javax.servlet.http package

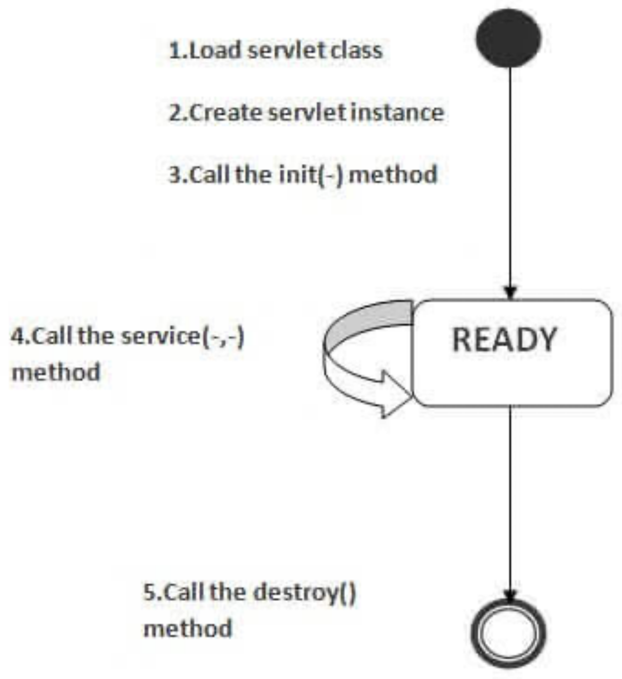
There are many classes in javax.servlet.http package. They are as follows:

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

Life Cycle of a Servlet

A servlet life cycle can be defined as the entire process from its creation till the destruction.

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 for creating and running a servlet

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

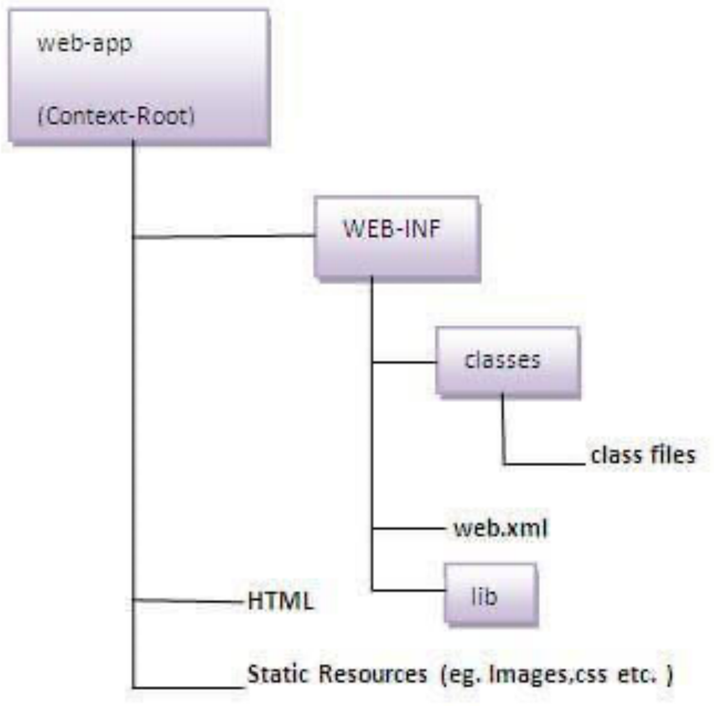
The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

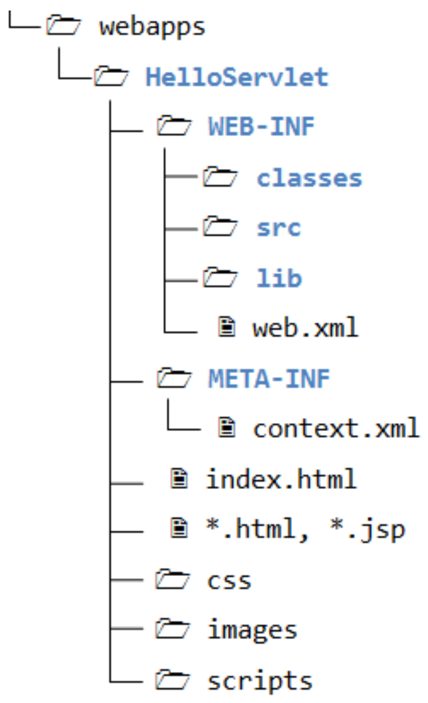
Following are the steps to create a servlet using Apache Tomcat Server:

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





```
└─ webapps
    └─ HelloServlet
        ├─ WEB-INF
        │   ├─ classes
        │   ├─ src
        │   ├─ lib
        │   └─ web.xml
        ├─ META-INF
        │   └─ context.xml
        ├─ index.html
        ├─ *.html, *.jsp
        ├─ css
        ├─ images
        └─ scripts
```

2) Create a Servlet

There are three ways to create the servlet.

- By implementing the Servlet interface
- By inheriting the GenericServlet class
- By inheriting the HttpServlet class

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

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{

	public void doGet(HttpServletRequest req,HttpServletResponse res)
		throws ServletException,IOException
	{
		res.setContentType("text/html");//setting the content type
		PrintWriter out=res.getWriter();//get the stream to write
		the data

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

	}
}
```

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

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

</web-app>
```

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

5) Start the Server and deploy the project

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

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

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

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

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

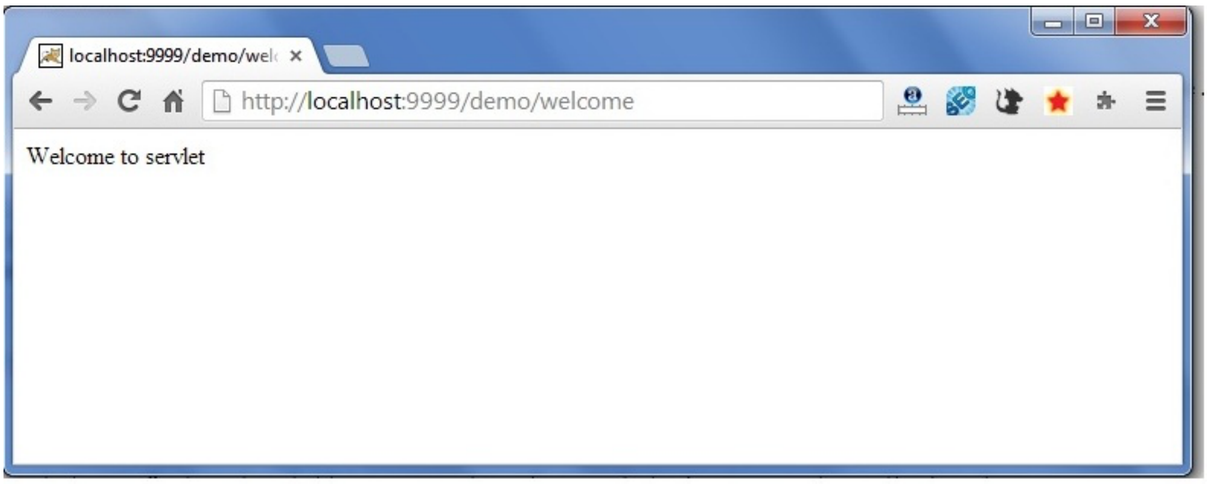
```
projectfolder> jar cvf myproject.war *
```

Creating war file has an advantage that moving the project from one location to another takes less time.

6) Accessing a servlet

Open browser and write ***http://hostname:portno/contextroot/urlpatternofservlet***. For example:

http://localhost:9999/demo/welcome



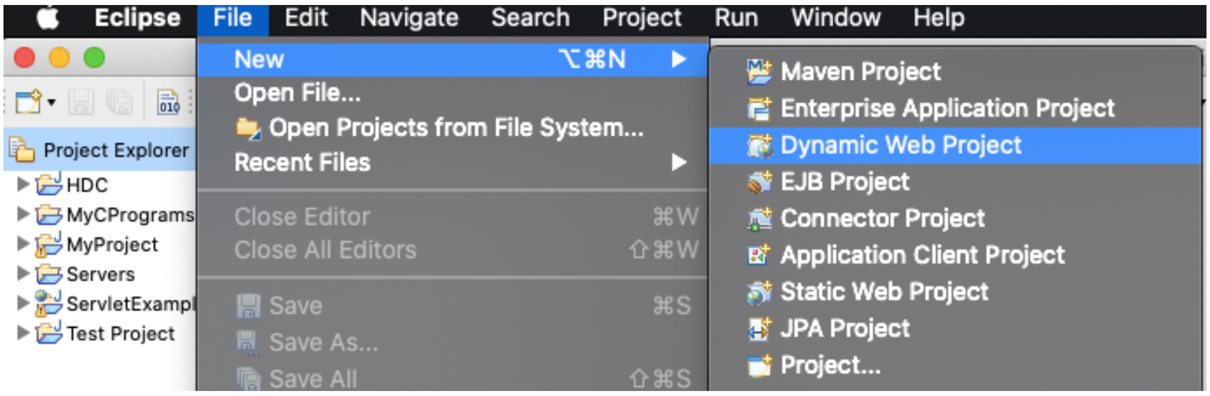
Welcome to servlet

Creating Servlet in Eclipse IDE

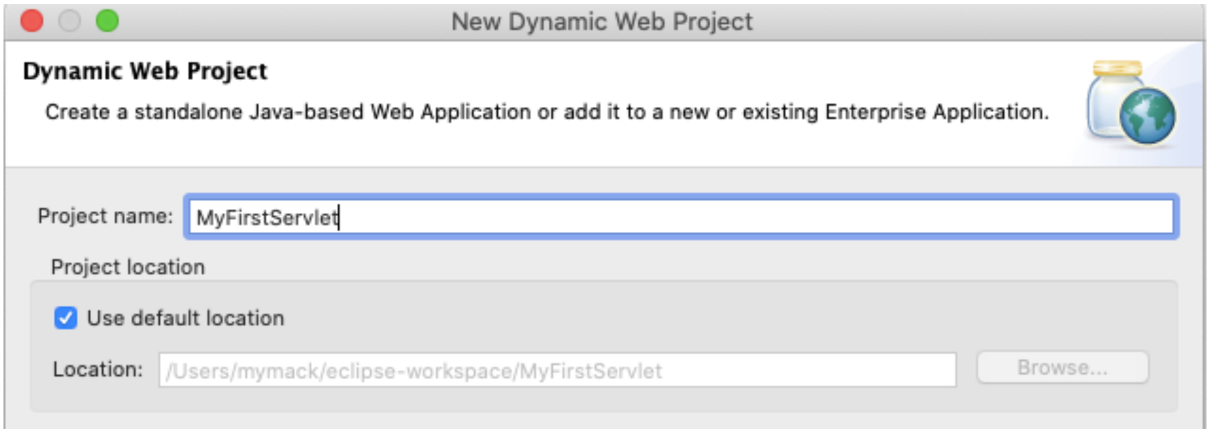
Firstly, you need to download and install Eclipse Enterprise Edition (EE) for web application development.

Step 1: Create a Project:

Lets create a Servlet application in Eclipse. Open Eclipse and then click File ❯ New ❯ Click Dynamic Web Project.



Give Project name and click Next.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Java-based Web Application or add it to a new or existing Enterprise Application.

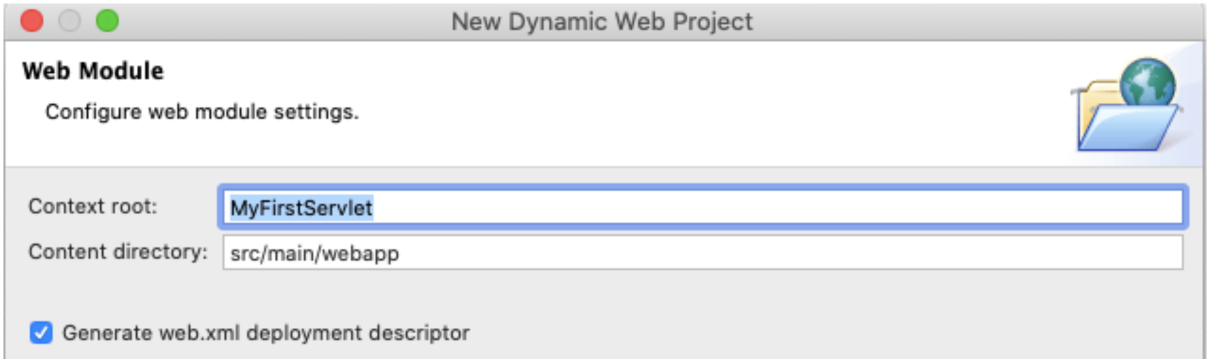
Project name: MyFirstServlet

Project location

✓ Use default location

Location: /Users/mymack/eclipse-workspace/MyFirstServlet

Tick the checkbox that says **Generate web.xml deployment descriptor**



New Dynamic Web Project

Web Module

Configure web module settings.

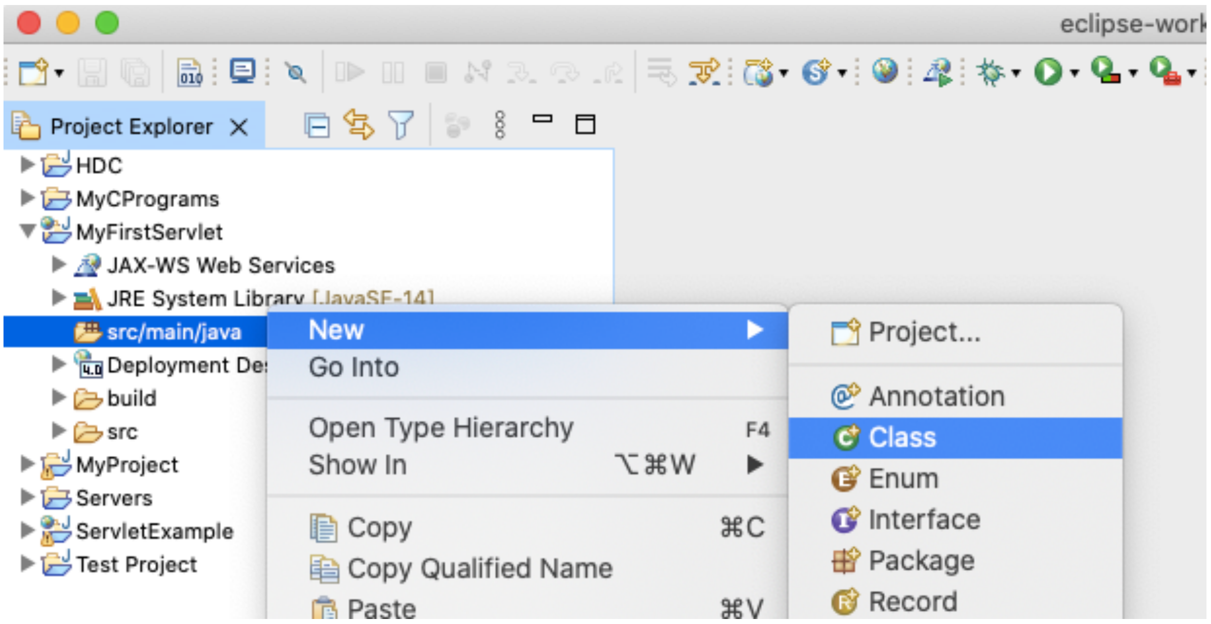
Context root: MyFirstServlet

Content directory: src/main/webapp

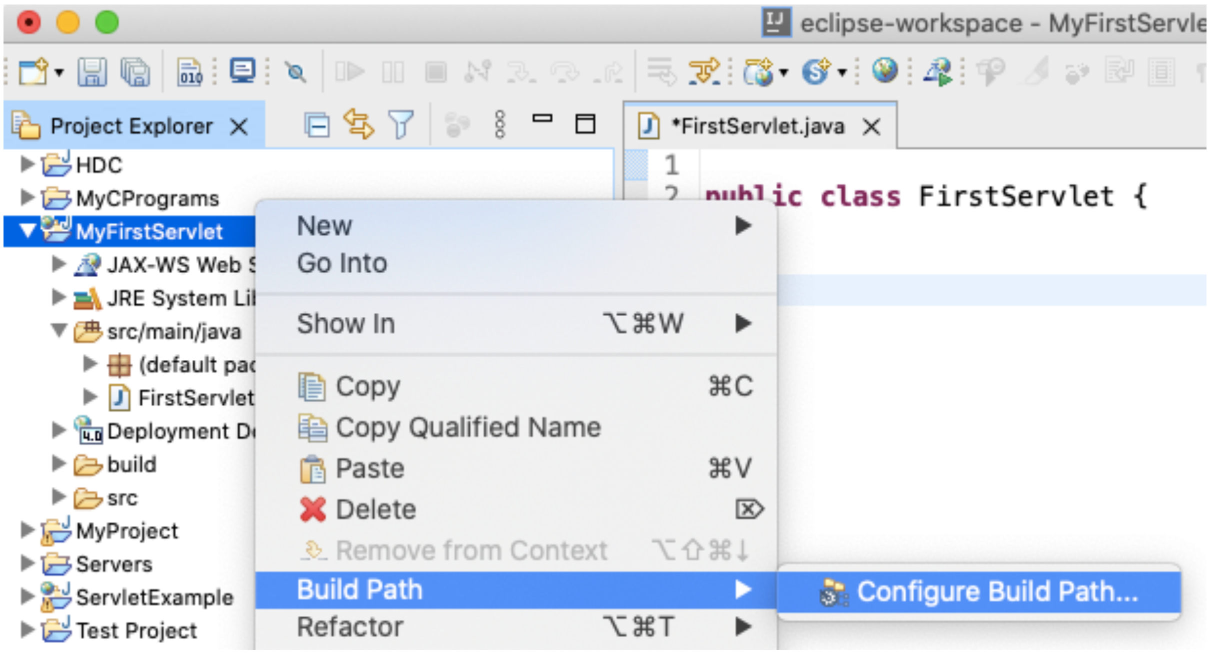
✓ Generate web.xml deployment descriptor

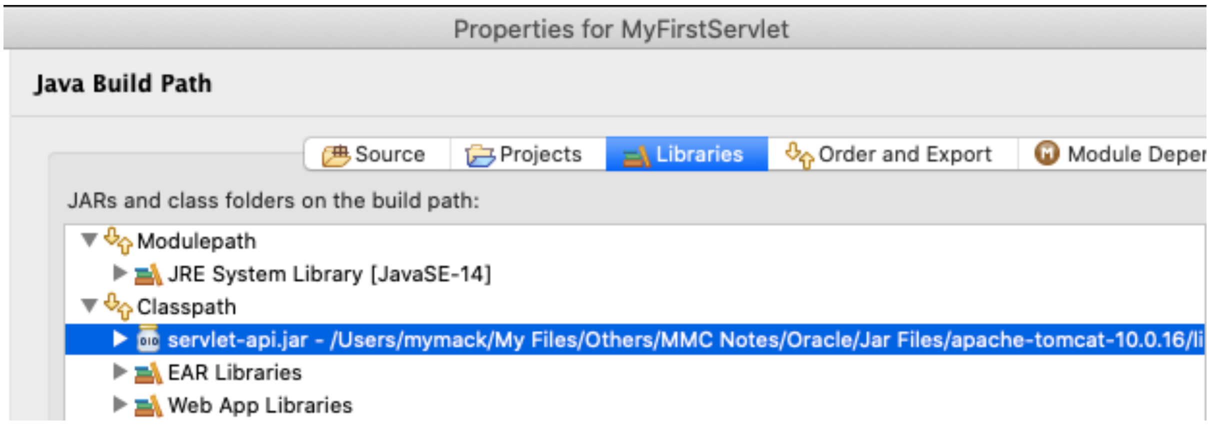
Step 2: Create a Servlet class:

We are creating a Http Servlet by extending HttpServlet class. Right click on the src folder and create a new class file, name the file as **FirstServlet**.



Now it's time to add servlet-api.jar file to your project.




Properties for MyFirstServlet

Java Build Path

JARs and class folders on the build path:

```
▼ Modulepath
    ▶ JRE System Library [JavaSE-14]
▼ Classpath
    ▶ servlet-api.jar - /Users/mymack/My Files/Others/MMC Notes/Oracle/Jar Files/apache-tomcat-10.0.16/li
    ▶ EAR Libraries
    ▶ Web App Libraries
```

You can get this jar file under apache-tomacat/lib folder.

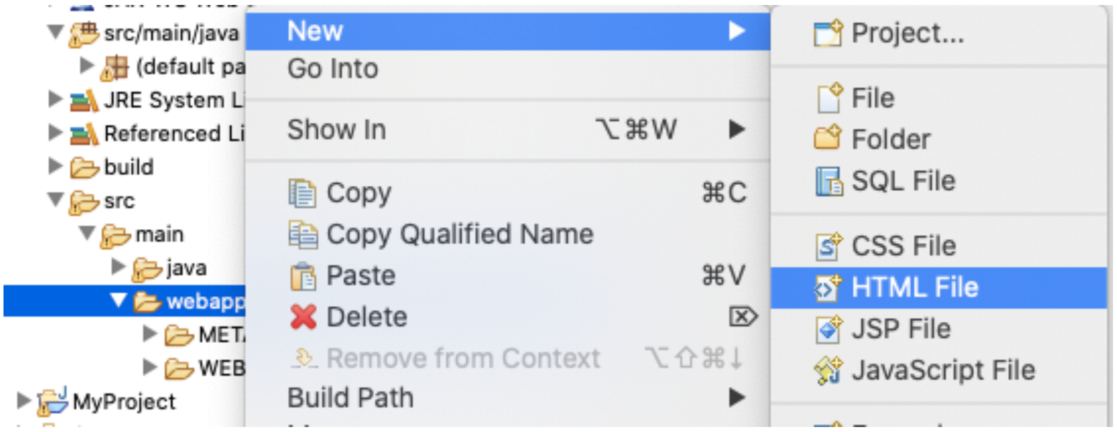
Now, it's a time to write your servlet program.

FirstServlet.java

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

//creating HttpServlet by extending HttpServlet
public class FirstServlet extends HttpServlet {
      private String msg;

      //init method of servlet
      public void init() throws ServletException {
            msg="This is my first servlet program";
      }

      //service method of servlet
      public void doGet(HttpServletRequest req,HttpServletResponse res)
            throws ServletException,IOException{

            //setting the content type of webpage
            res.setContentType("text/html");

            //writing a message in webpage
            PrintWriter out=res.getWriter();
            out.println("<h1>" +msg+ "</h1>");
      }

      public void destroy() {
            //code on servlet destroy
      }
}
```

Step 3: Create an html page to call the servlet class on a webpage

We are creating an html file that **would call the servlet once we click on the link** on web page. Create this file in WebContent folder. **The path of the file should look like this: WebContent/index.html**



index.html

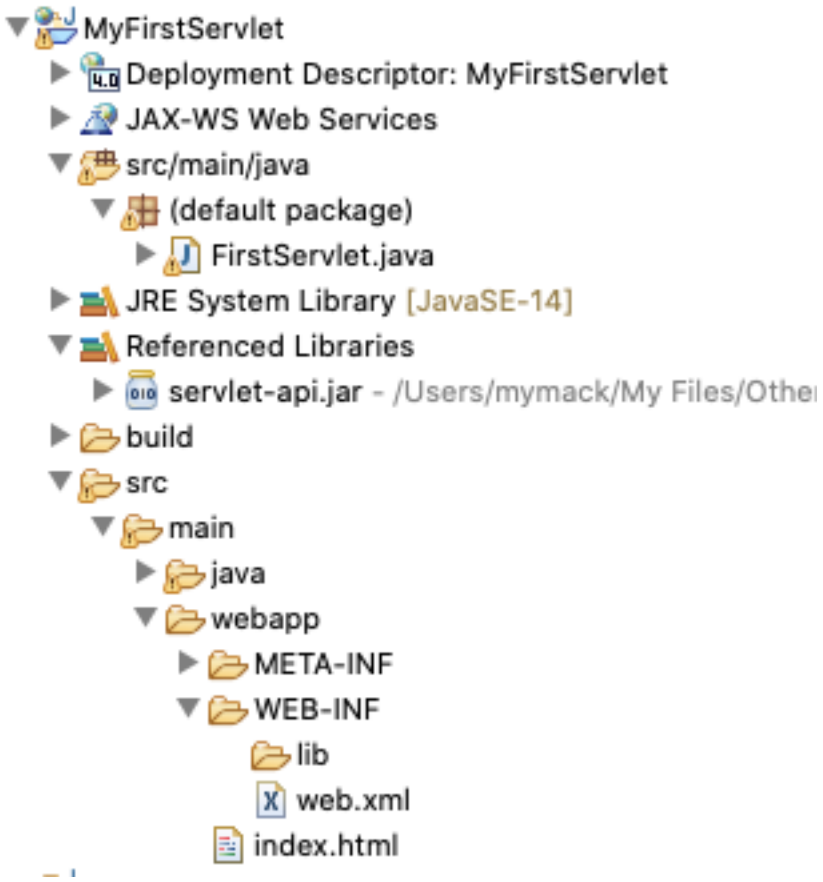
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
     <a href="myservlet">Click to call Servlet</a>
</body>
</html>
```

Edit web.xml file

This file can be found at this path WebContent/WEB-INF/web.xml. In this file we will map the Servlet with the specific URL. Since we are calling welcome page upon clicking the link on index.html page so we are mapping the welcome page to the Servlet class we created above.

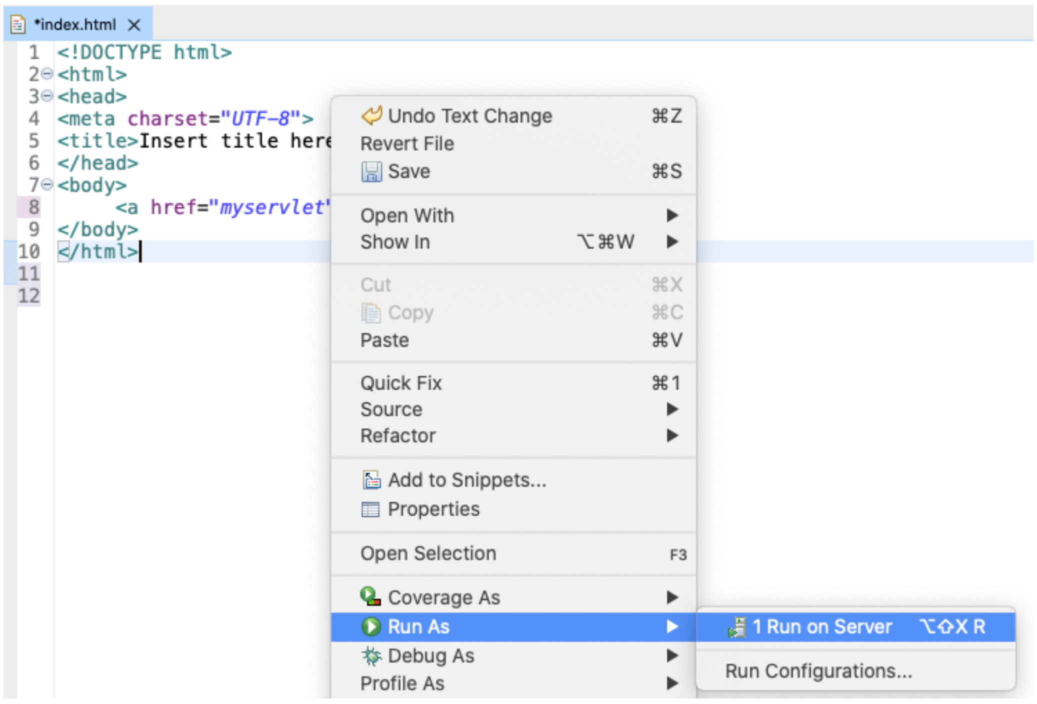
```
<servlet>
     <servlet-name>FirstServlet</servlet-name>
     <servlet-class>FirstServlet</servlet-class>
  </servlet>

  <servlet-mapping>
     <servlet-name>FirstServlet</servlet-name>
     <url-pattern>/myservlet</url-pattern>
  </servlet-mapping>
```

Final Project Structure



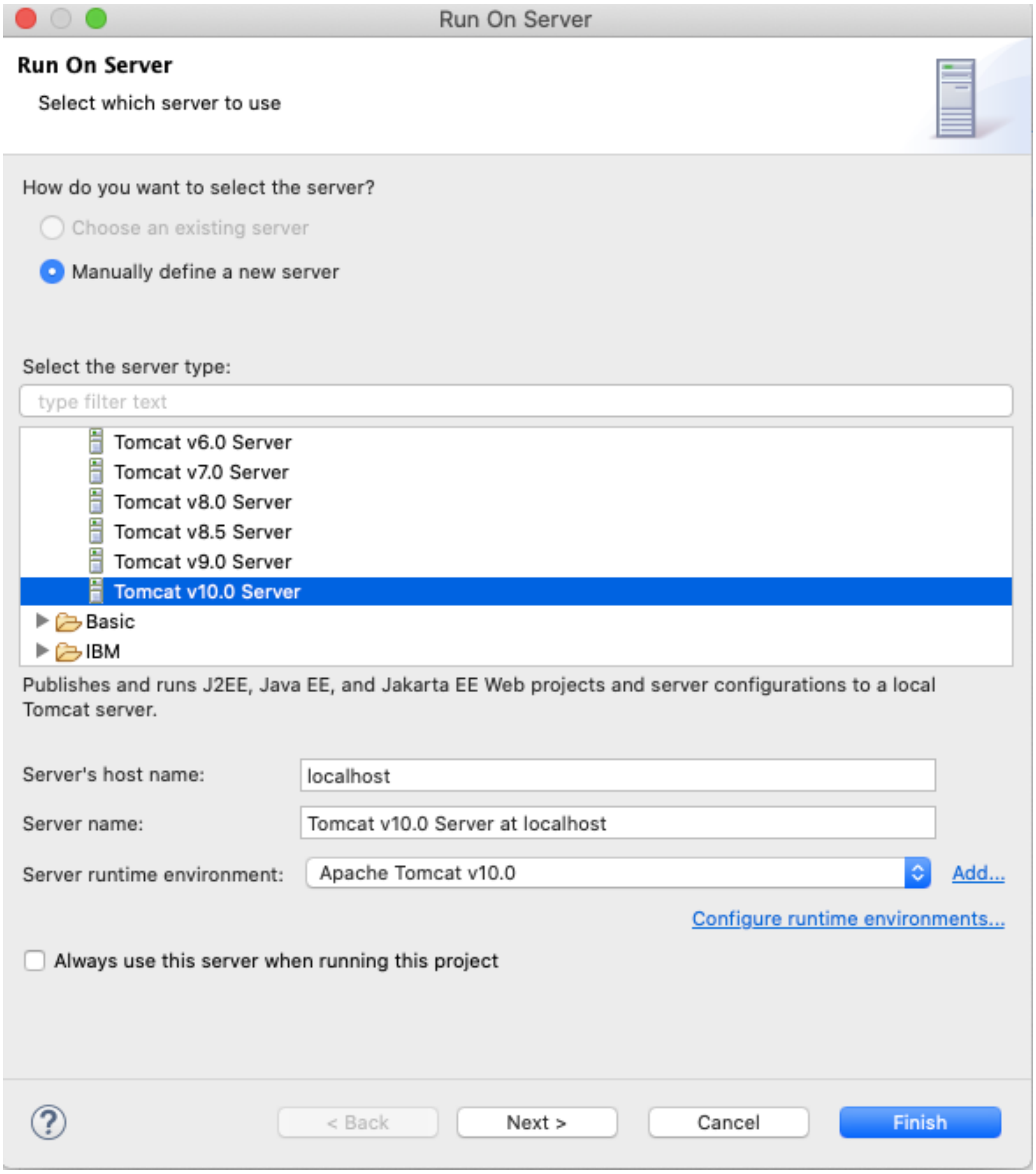
Step 4: Run the project

Right click on the index.html, run on server.



```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here
</head>
<body>
    <a href="myservlet"
</body>
</html>
```

Configure a server to run your application



Run On Server

Run On Server

Select which server to use

How do you want to select the server?

( ) Choose an existing server

(●) Manually define a new server

Select the server type:

type filter text

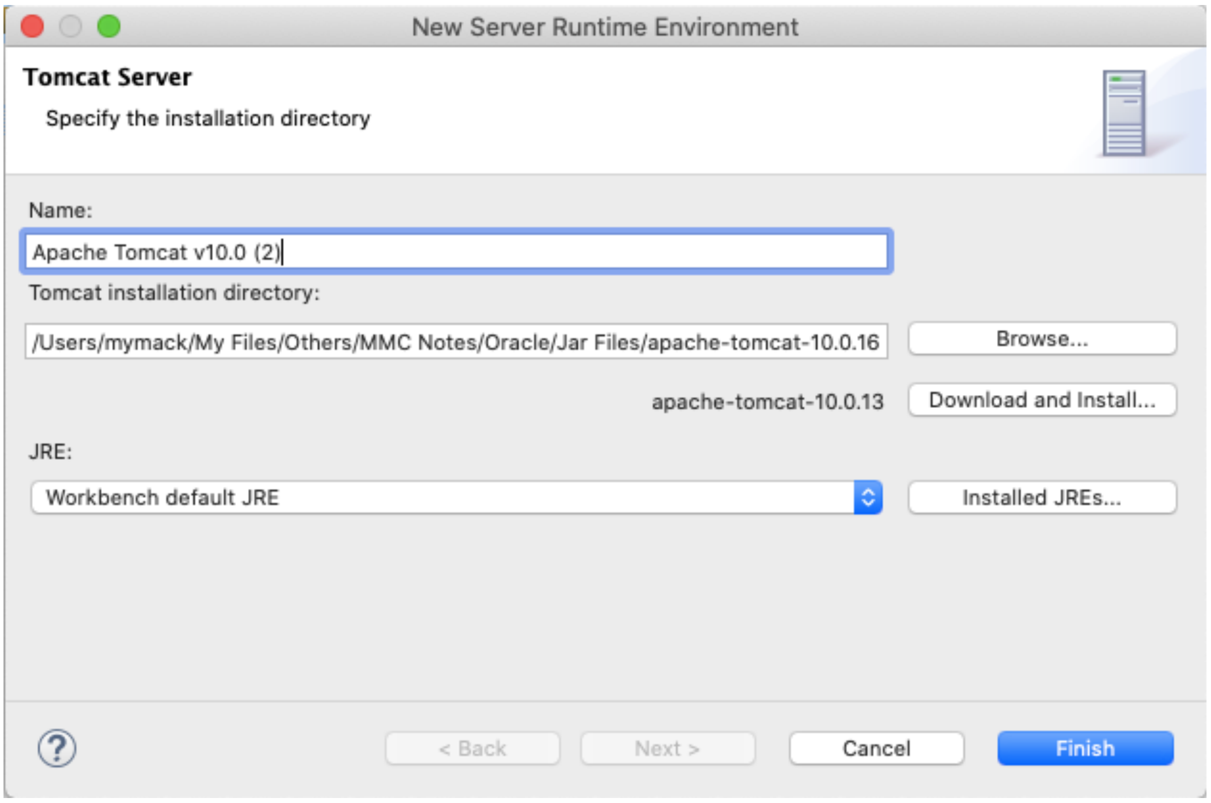
- Tomcat v6.0 Server
- Tomcat v7.0 Server
- Tomcat v8.0 Server
- Tomcat v8.5 Server
- Tomcat v9.0 Server
- **Tomcat v10.0 Server**
- Basic
- IBM

Publishes and runs J2EE, Java EE, and Jakarta EE Web projects and server configurations to a local Tomcat server.

Server's host name:	localhost
Server name:	Tomcat v10.0 Server at localhost
Server runtime environment:	Apache Tomcat v10.0

[ ] Always use this server when running this project

Click Add Button to setup Server runtime environment.


New Server Runtime Environment

Tomcat Server

Specify the installation directory

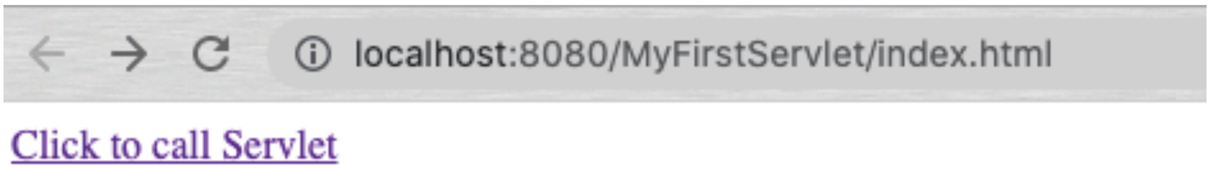
Name: Apache Tomcat v10.0 (2)

Tomcat installation directory: /Users/mymack/My Files/Others/MMC Notes/Oracle/Jar Files/apache-tomcat-10.0.16

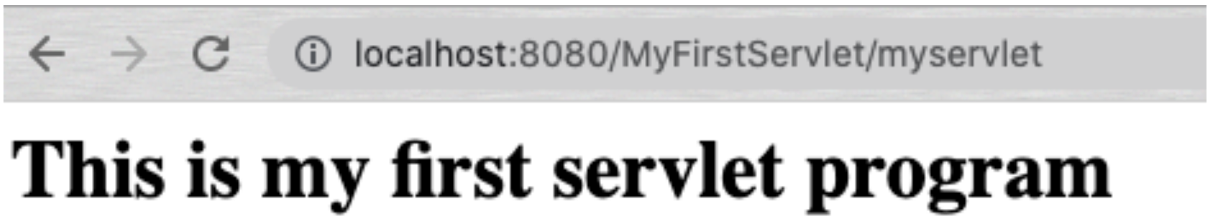
apache-tomcat-10.0.13

JRE: Workbench default JRE

Output:


localhost:8080/MyFirstServlet/index.html

Click to call Servlet


localhost:8080/MyFirstServlet/myservlet

This is my first servlet program

Servlet Interface

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.**

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.

Example of Servlet Interface

```java
import java.io.*;
import javax.servlet.*;
public class DemoServlet implements Servlet{
   ServletConfig config=null;
   public void init(ServletConfig config){
      this.config=config;
      System.out.println("Initialization complete");
   }

   public void service(ServletRequest req,ServletResponse res)
   throws IOException,ServletException{
       res.setContentType("text/html");
       PrintWriter pwriter=res.getWriter();
       pwriter.print("<html>");
       pwriter.print("<body>");
       pwriter.print("<h1>Servlet Example Program</h1>");
       pwriter.print("</body>");
       pwriter.print("</html>");
   }
   public void destroy(){
       System.out.println("servlet life cycle finished");
   }
   public ServletConfig getServletConfig(){
       return config;
```

```
    }
    public String getServletInfo(){
        return "Servlet Test";
    }
}
```

Generic Servlet Class

A generic servlet is a protocol independent Servlet that should always override the service() method to handle the client request. The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

Pros of using Generic Servlet:

- Generic Servlet is easier to write
- Has simple lifecycle methods
- To write Generic Servlet you just need to extend javax.servlet.GenericServlet and override the service() method.

Cons of using Generic Servlet:

- Working with Generic Servlet is not that easy because we don't have convenience methods such as doGet(), doPost(), doHead() etc in Generic Servlet that we can use in Http Servlet.

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.

Example of Generic Servlet Class:

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

public class ExampleGeneric extends GenericServlet{

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

              res.setContentType("text/html");
              PrintWriter pwriter=res.getWriter();
              pwriter.print("<html>");
              pwriter.print("<body>");
              pwriter.print("<h2>Generic Servlet Example</h2>");
              pwriter.print("</body>");
              pwriter.print("</html>");
       }
 }
```

Since Generic Servlet is a class, so we don't need to write all the methods of this class like in Servlet Interface.

Reading Servlet Parameters (Handling form data)

Servlets handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

Reading data with GET Request

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the **?** (question mark) symbol as follows −

http://www.test.com/hello?key1 = value1&key2 = value2

The GET method is the default method to pass information from browser to web server. Never use the GET method if you have password or other sensitive information to pass to the server. **The GET method has size limitation: only 1024 characters can be used in a request string.**

Example is shown below:

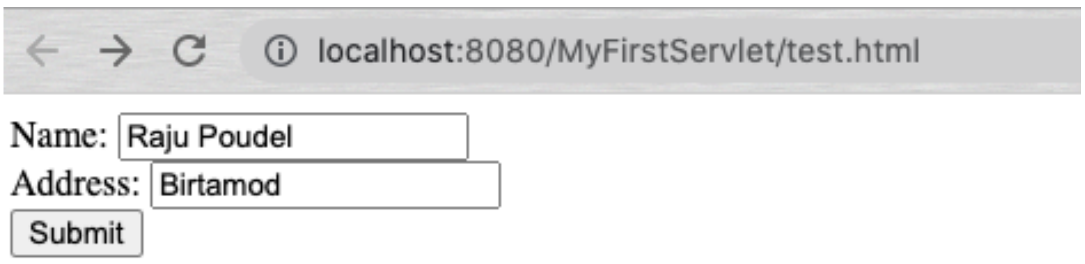
test.html

```
<html>
<body>
   <form method="GET" action="FirstServlet">
           Name: <input type="text" name="name"/>
           <br/>
           Address: <input type="text" name="address"/>
           <br/>
           <input type="submit" value="Submit"/>
   </form>
</body>
</html>
```

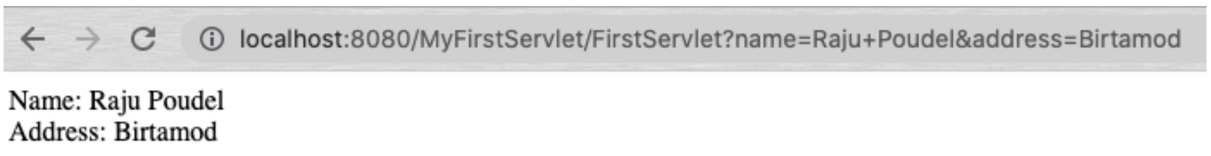
FirstServlet.java

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

public class FirstServlet extends HttpServlet {

     //doGet handles get request
public void doGet(HttpServletRequest req, HttpServletResponse res)
           throws ServletException,IOException{
           //reading form data
           String name=req.getParameter("name");
           String address=req.getParameter("address");

           //setting content type
           res.setContentType("text/html");
           PrintWriter out=res.getWriter();
           //displaying data in html
           out.println("<html>");
           out.println("<body><p> Name: "+name+"<br>");
           out.println("Address: "+address+"</p></body></html>");

     }
}
```

Output:



Name: Raju Poudel
Address: Birtamod


```
localhost:8080/MyFirstServlet/FirstServlet?name=Raju+Poudel&address=Birtamod

Name: Raju Poudel
Address: Birtamod
```

Reading data with POST Request

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

test.html

```
<html>
<body>
   <form method="POST" action="FirstServlet">
           Name: <input type="text" name="name"/>
           <br/>
           Address: <input type="text" name="address"/>
           <br/>
           <input type="submit" value="Submit"/>
   </form>
</body>
</html>
```

FirstServlet.java

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

public class FirstServlet extends HttpServlet {
      //doPost handles post request
public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException,IOException{
           //reading form data
           String name=req.getParameter("name");
           String address=req.getParameter("address");

           //setting content type
           res.setContentType("text/html");
           PrintWriter out=res.getWriter();
           //displaying data in html
           out.println("<html>");
           out.println("<body><p> Name: "+name+"<br>");
           out.println("Address: "+address+"</p></body></html>");
      }
}
```

Reading All Form Parameters

getParameterNames() method of HttpServletRequest is used to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.
Once we have an Enumeration, we can loop down the Enumeration in standard way by, using ***hasMoreElements()*** method to determine when to stop and using ***nextElement()*** method to get each parameter name.

Example is shown below:

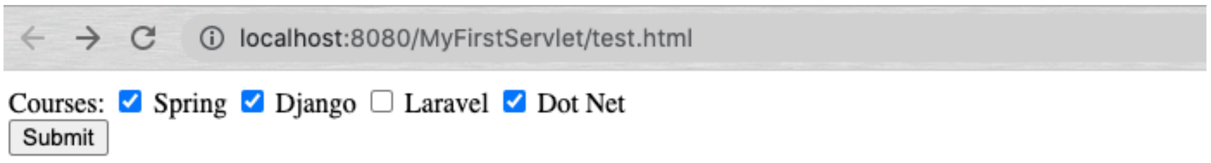
test.html

```
<html>
<body>
   <form method="POST" action="FirstServlet">
           Courses:
           <input type="checkbox" name="spring" value="Spring"
                 checked/> Spring
           <input type="checkbox" name="django" value="Django"/>
                 Django
           <input type="checkbox" name="laravel" value="Laravel"/>
                 Laravel
           <input type="checkbox" name="dotnet" value="Dot Net"/>
                 Dot Net
           <br>
           <input type="submit" value="Submit"/>
   </form>
</body>
</html>
```

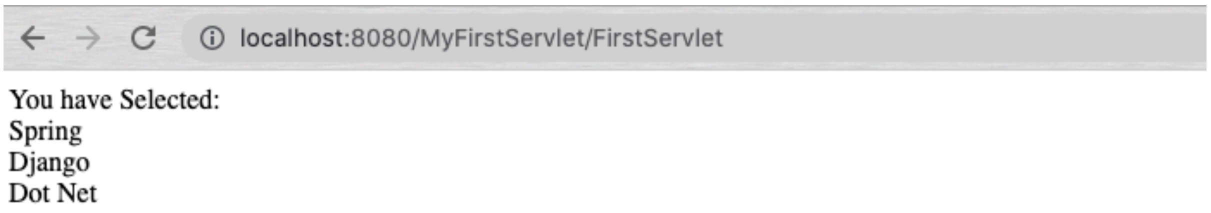
FirstServlet.java

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

public class FirstServlet extends HttpServlet {
      //doPost handles get request
    public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException,IOException{

            res.setContentType("text/html");
            PrintWriter out=res.getWriter();
            out.println("<html><body>");
            out.println("You have Selected: <br>");

            //getting all parameters at once
            Enumeration paramNames=req.getParameterNames();
            //looping to get all parameters
            while(paramNames.hasMoreElements()) {
                //getting single parameter
                String pname=(String) paramNames.nextElement();

            /*
                we can put all values in single array and access later
                String values[]=req.getParameterValues(pname);
            */

                //getting parameter value individually
                String value=req.getParameter(pname);
                //displaying value
                out.println(value+"<br>");
            }
            out.println("</body></html>");
        }
}
```

Output:


localhost:8080/MyFirstServlet/test.html

Courses: ☑ Spring ☑ Django ☐ Laravel ☑ Dot Net


localhost:8080/MyFirstServlet/FirstServlet

You have Selected:
Spring
Django
Dot Net

Client HTTP Request

When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a **part of header of HTTP request**. **Following is the important header information which comes from browser side and you would use very frequently in web programming –**

Sr.No.	Header & Description
1	**Accept** This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or **image/jpeg** are the two most common possibilities.
2	**Accept-Charset** This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.

3	**Accept-Encoding** This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or **compress** are the two most common possibilities.
4	**Accept-Language** This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc
5	**Authorization** This header is used by clients to identify themselves when accessing password-protected Web pages.
6	**Connection** This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used.
7	**Content-Length** This header is applicable only to POST requests and gives the size of the POST data in bytes.
8	**Cookie** This header returns cookies to servers that previously sent them to the browser.
9	**Host** This header specifies the host and port as given in the original URL.
10	**If-Modified-Since** This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means **Not Modified** header if no newer result is available.
11	**If-Unmodified-Since** This header is the reverse of If-Modified-Since; it specifies that the operation should succeed only if the document is older than the specified date.
12	**Referer** This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referrer header when the browser requests Web page 2.
13	**User-Agent** This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

Methods to read HTTP Header

There are following methods which can be used to read HTTP header in your servlet program. These methods are available with *HttpServletRequest* object.

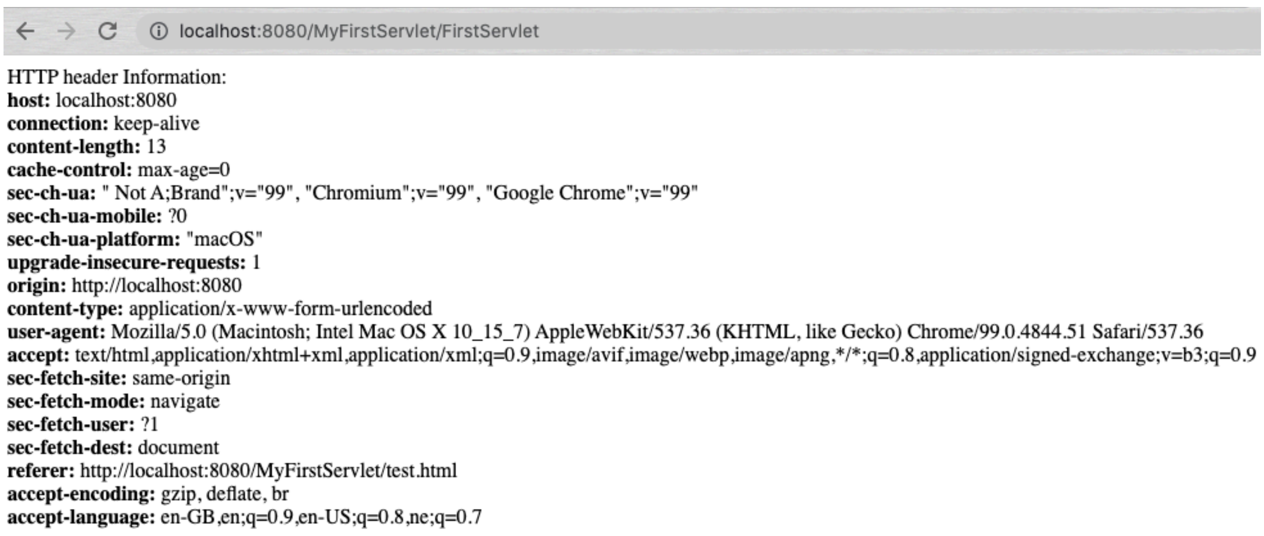
Sr.No.	Method & Description
1	**Cookie[] getCookies()** Returns an array containing all of the Cookie objects the client sent with this request.
2	**Enumeration getAttributeNames()** Returns an Enumeration containing the names of the attributes available to this request.
3	**Enumeration getHeaderNames()** Returns an enumeration of all the header names this request contains.
4	**Enumeration getParameterNames()** Returns an Enumeration of String objects containing the names of the parameters contained in this request
5	**HttpSession getSession()** Returns the current session associated with this request, or if the request does not have a session, creates one.
6	**HttpSession getSession(boolean create)** Returns the current HttpSession associated with this request or, if if there is no current session and value of create is true, returns a new session.
7	**Locale getLocale()** Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.
8	**Object getAttribute(String name)** Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
9	**ServletInputStream getInputStream()** Retrieves the body of the request as binary data using a ServletInputStream.
10	**String getAuthType()** Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.
11	**String getCharacterEncoding()** Returns the name of the character encoding used in the body of this request.
12	**String getContentType()** Returns the MIME type of the body of the request, or null if the type is not known.
13	**String getContextPath()** Returns the portion of the request URI that indicates the context of the request.
14	**String getHeader(String name)** Returns the value of the specified request header as a String.
15	**String getMethod()** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
16	**String getParameter(String name)**

	Returns the value of a request parameter as a String, or null if the parameter does not exist.
17	**String getPathInfo()** Returns any extra path information associated with the URL the client sent when it made this request
18	**String getProtocol()** Returns the name and version of the protocol the request.
19	**String getQueryString()** Returns the query string that is contained in the request URL after the path.
20	**String getRemoteAddr()** Returns the Internet Protocol (IP) address of the client that sent the request.
21	**String getRemoteHost()** Returns the fully qualified name of the client that sent the request.
22	**String getRemoteUser()** Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
23	**String getRequestURI()** Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
24	**String getRequestedSessionId()** Returns the session ID specified by the client.
25	**String getServletPath()** Returns the part of this request's URL that calls the JSP.
26	**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.
27	**boolean isSecure()** Returns a Boolean indicating whether this request was made using a secure channel, such as HTTPS.
28	**int getContentLength()** Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
29	**int getIntHeader(String name)** Returns the value of the specified request header as an int.
30	**int getServerPort()** Returns the port number on which this request was received.

Example to Display Header information

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws IOException, ServletException
  {
     res.setContentType("text/html");
     PrintWriter out = res.getWriter();
     out.println("HTTP header Information:<br>");
     Enumeration en = req.getHeaderNames();
     while (en.hasMoreElements()) {
        String hName = (String) en.nextElement();
        String hValue = req.getHeader(hName);
        out.println("<b>"+hName+": </b>"
             +hValue + "<br>");
     }
  }
```

Output:



localhost:8080/MyFirstServlet/FirstServlet

```
HTTP header Information:
host: localhost:8080
connection: keep-alive
content-length: 13
cache-control: max-age=0
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
upgrade-insecure-requests: 1
origin: http://localhost:8080
content-type: application/x-www-form-urlencoded
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
sec-fetch-site: same-origin
sec-fetch-mode: navigate
sec-fetch-user: ?1
sec-fetch-dest: document
referer: http://localhost:8080/MyFirstServlet/test.html
accept-encoding: gzip, deflate, br
accept-language: en-GB,en;q=0.9,en-US;q=0.8,ne;q=0.7
```

Server HTTP Response

When a Web server responds to an HTTP request, the response typically consists of a status line, some response headers, a blank line, and the document. **A typical response looks like this –**

```
HTTP/1.1 200 OK
Content-Type: text/html
Header2: ...
...
HeaderN: ...
   (Blank Line)
<!doctype ...>
<html>
   <head>...</head>
   <body>
      ...
   </body>
</html>
```

The status line consists of the HTTP version (HTTP/1.1 in the example), a status code (200 in the example), and a very short message corresponding to the status code (OK in the example).

Following is a summary of the most useful HTTP 1.1 response headers which go back to the browser from web server side and you would use them very frequently in web programming –

Sr.No.	Header & Description
1	**Allow** This header specifies the request methods (GET, POST, etc.) that the server supports.
2	**Cache-Control** This header specifies the circumstances in which the response document can safely be cached. It can have values **public**, **private** or **no-cache** etc. Public means document is cacheable, Private means document is for a single user and can only be stored in private (non-shared) caches and nocache means document should never be cached.
3	**Connection** This header instructs the browser whether to use persistent in HTTP connections or not. A value of **close** instructs the browser not to use persistent HTTP connections and **keepalive** means using persistent connections.
4	**Content-Disposition** This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
5	**Content-Encoding** This header specifies the way in which the page was encoded during transmission.
6	**Content-Language** This header signifies the language in which the document is written. For example en, en-us, ru, etc
7	**Content-Length** This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
8	**Content-Type** This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
9	**Expires** This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.
10	**Last-Modified**

	This header indicates when the document was last changed. The client can then cache the document and supply a date by an **If-Modified-Since** request header in later requests.
11	**Location** This header should be included with all responses that have a status code in the 300s. This notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document.
12	**Refresh** This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
13	**Retry-After** This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
14	**Set-Cookie** This header specifies a cookie associated with the page.

Methods to Set HTTP Response Header

There are following methods which can be used to set HTTP response header in your servlet program. These methods are available with *HttpServletResponse* object.

Sr.No.	Method & Description
1	**String encodeRedirectURL(String url)** Encodes the specified URL for use in the sendRedirect method or, if encoding is not needed, returns the URL unchanged.
2	**String encodeURL(String url)** Encodes the specified URL by including the session ID in it, or, if encoding is not needed, returns the URL unchanged.
3	**boolean containsHeader(String name)** Returns a Boolean indicating whether the named response header has already been set.
4	**boolean isCommitted()** Returns a Boolean indicating if the response has been committed.
5	**void addCookie(Cookie cookie)** Adds the specified cookie to the response.
6	**void addDateHeader(String name, long date)** Adds a response header with the given name and date-value.
7	**void addHeader(String name, String value)** Adds a response header with the given name and value.
8	**void addIntHeader(String name, int value)** Adds a response header with the given name and integer value.
9	**void flushBuffer()** Forces any content in the buffer to be written to the client.
10	**void reset()** Clears any data that exists in the buffer as well as the status code and headers.
11	**void resetBuffer()**

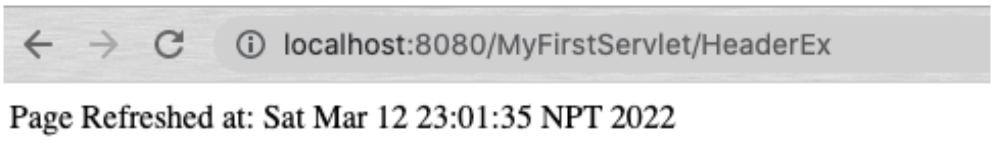
	Clears the content of the underlying buffer in the response without clearing headers or status code.
12	**void sendError(int sc)** Sends an error response to the client using the specified status code and clearing the buffer.
13	**void sendError(int sc, String msg)** Sends an error response to the client using the specified status.
14	**void sendRedirect(String location)** Sends a temporary redirect response to the client using the specified redirect location URL.
15	**void setBufferSize(int size)** Sets the preferred buffer size for the body of the response.
16	**void setCharacterEncoding(String charset)** Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
17	**void setContentLength(int len)** Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header.
18	**void setContentType(String type)** Sets the content type of the response being sent to the client, if the response has not been committed yet.
19	**void setDateHeader(String name, long date)** Sets a response header with the given name and date-value.
20	**void setHeader(String name, String value)** Sets a response header with the given name and value.
21	**void setIntHeader(String name, int value)** Sets a response header with the given name and integer value
22	**void setLocale(Locale loc)** Sets the locale of the response, if the response has not been committed yet.
23	**void setStatus(int sc)** Sets the status code for this response

HTTP Header Response Example

You already have seen setContentType() method working in previous examples and following example would also use same method, additionally we would use **setIntHeader()** method to set **Refresh** header.

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

@WebServlet("/HeaderEx")
public class HeaderEx extends HttpServlet {

protected void doGet(HttpServletRequest req, HttpServletResponse res)
                 throws ServletException, IOException {

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

           //refreshing page in every 5 seconds
           res.setIntHeader("Refresh", 5);

           //getting current date and time
           Date date=new Date();

           out.println("Page Refreshed at: "+date);
      }
}
```

Output:



localhost:8080/MyFirstServlet/HeaderEx

```
Page Refreshed at: Sat Mar 12 23:01:35 NPT 2022
```

Request Dispatcher

The RequestDispatcher interface defines an object that receives the request from client and dispatches it to the resource(such as servlet, JSP, HTML file).

This interface has following two methods:

- **public void forward(ServletRequest request, ServletResponse response)**: It forwards the request from one servlet to another resource (such as servlet, JSP, HTML file).
- **public void include(ServletRequest request, ServletResponse response)**: It includes the content of the resource(such as servlet, JSP, HTML file) in the response.

Difference between forward() vs include() method

To understand the difference between these two methods, lets take an example: Suppose you have two pages X and Y. In page X you have an include tag, this means that the control will be in the page X till it encounters include tag, after that the control will be transferred to page Y. At the end of the processing of page Y, the control will return back to the page X starting just after the include tag and remain in X till the end.
In this case the final response to the client will be send by page X.

Now, we are taking the same example with forward. We have same pages X and Y. In page X, we have forward tag. In this case the control will be in page X till it encounters forward, after this the control will be transferred to page Y. The main difference here is that the control will not return back to X, it will be in page Y till the end of it.
In this case the final response to the client will be send by page Y.

Example of Request Dispatcher:

index.html

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

Login.java

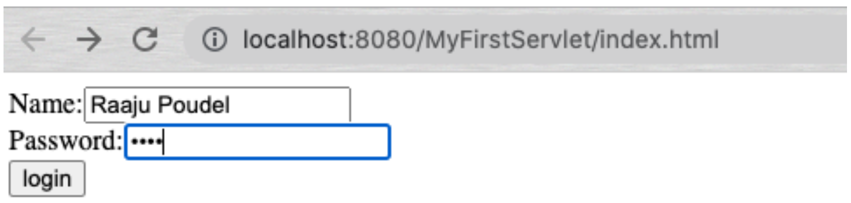
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Login extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse r
   esponse)  throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter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

```
        String n=request.getParameter("userName");
        String p=request.getParameter("userPass");
        if(p.equals("raaju")){
               RequestDispatcher rd=request.
                        getRequestDispatcher("WelcomeServlet");
            rd.forward(request, response);
        }
        else{
            out.print("Sorry UserName or Password Error!");
            RequestDispatcher rd=request.
                  getRequestDispatcher("/index.html");
            rd.include(request, response);
            }
        }
    }
```

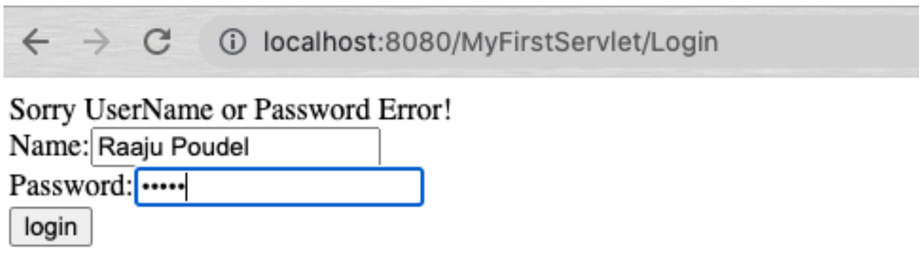
<u>WelcomeServlet.java</u>

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse
           response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    }
}
```

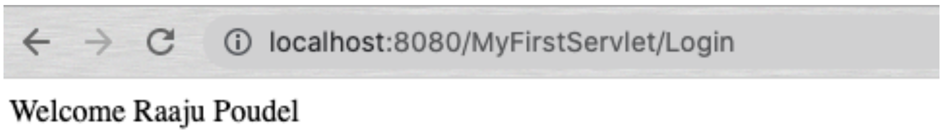
<u>Output:</u>


localhost:8080/MyFirstServlet/index.html

Name: Raaju Poudel
Password: ••••
login


localhost:8080/MyFirstServlet/Login

Sorry UserName or Password Error!
Name: Raaju Poudel
Password: •••••
login

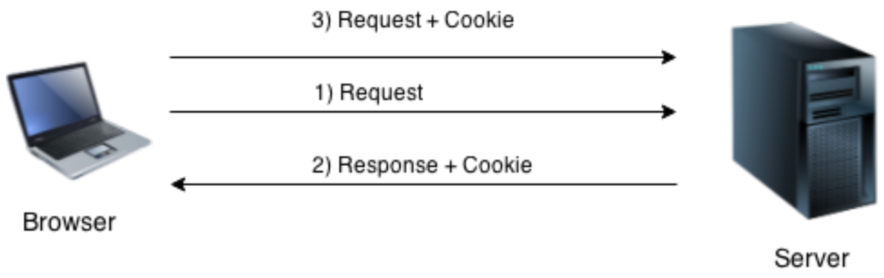

localhost:8080/MyFirstServlet/Login

Welcome Raaju Poudel

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

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



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

1) Session Cookies:

- **Session cookies do not have expiration time**. It lives in the **browser** memory. As soon as the web browser is closed this cookie gets destroyed.

2) Persistent Cookies:

- **Unlike Session cookies they have expiration time**, they are stored in the user hard drive and gets **destroyed based on the expiry time**.

Advantage of Cookies

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

Disadvantage of Cookies

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

Sending Cookies to a client:

Here are steps for sending cookie to the client:

1. Create a Cookie object.
2. Set the maximum Age.
3. Place the Cookie in HTTP response header.

1) Create a Cookie object:

```
        Cookie c = new Cookie("username","raaju");
```

2) Set the maximum Age:

By using **setMaxAge ()** method we can set the maximum age for the particular cookie in seconds.

```
        c.setMaxAge(1800);
```

3) Place the Cookie in HTTP response header:

We can send the cookie to the client browser through **response.addCookie()** method.

```
        Response.addCookie(c);
```

Reading cookies values:

```
Cookie c[]=request.getCookies();
//c.length gives the cookie count
for(int i=0;i<c.length;i++){
        out.print("Name: "+c[i].getName()+" & Value: "+c[i].getValue());
}
```

Example of Cookie in Servlet:

Here we're create **two servlets**, one for setting a cookie and another for accessing and displaying cookie values.

index.html

```
<html>
<body>
     <a href="first">Set Cookie</a>
     <a href="second">Display Cookie</a>
</body>
</html>
```

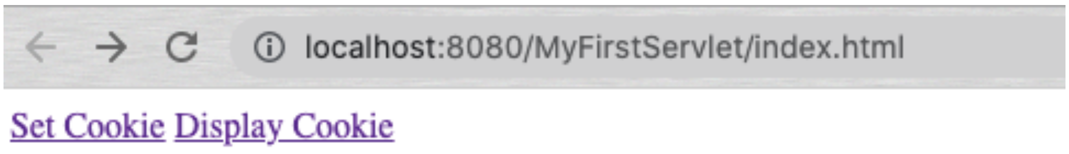
FirstServlet.java

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

@WebServlet("/first")
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException,IOException{
           //setting a cookie values
           Cookie c1=new Cookie("username","raaju");
           Cookie c2=new Cookie("password","admin");

           //adding a cookie in response
           res.addCookie(c1);
           res.addCookie(c2);

           res.setContentType("text/html");
           PrintWriter out=res.getWriter();
           out.println("Cookie Added Successfully!");
      }
}
```

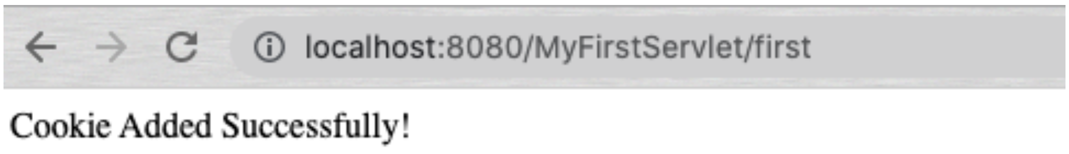
SecondServlet.java

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

@WebServlet("/second")
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException,IOException{

           //accessing cookie values
           Cookie c[]=req.getCookies();

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

           out.println("Username: "+c[0].getValue()+"<br>");
           out.println("Password: "+c[1].getValue());

      }
}
```

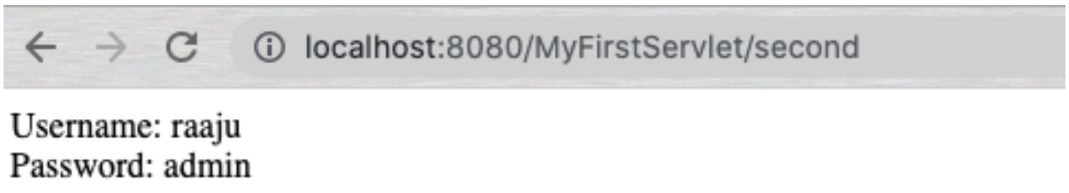
Output:


localhost:8080/MyFirstServlet/index.html

Set Cookie Display Cookie


localhost:8080/MyFirstServlet/first

Cookie Added Successfully!


localhost:8080/MyFirstServlet/second

```
Username: raaju
Password: admin
```

Methods of Cookie class

- **public void setComment(String purpose)**: This method is used for setting up comments in the cookie. This is basically used for describing the purpose of the cookie.
- **public String getComment()**: Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
- **public void setMaxAge(int expiry)**: Sets the maximum age of the cookie in seconds.
- **public int getMaxAge()**: Gets the maximum age in seconds of this Cookie. By default, -1 is returned, which indicates that the cookie will persist until browser shutdown.
- **public String getName()**: Returns the name of the cookie. The name cannot be changed after creation.
- **public void setValue(String newValue)**: Assigns a new value to this Cookie.
- **public String getValue()**: Gets the current value of this Cookie.

Session Management in Servlet

The HttpSession object is used for session management. A session contains information specific to a particular user across the whole application. When a user enters into a website (or an online application) for the first time HttpSession is obtained via **request.getSession(),** the user is given a unique ID to identify his session. This unique ID can be stored into a cookie or in a request parameter.
The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file(web.xml). The default timeout value is 30 minutes, this is used if you don't specify the value in tag. This means that when the user doesn't visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

This is how you create a HttpSession object.

```
protected void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
        HttpSession session = req.getSession();
}
```

You can store the user information into the session object by using setAttribute() method and later when needed this information can be fetched from the session.

```
session.setAttribute("username", "raaju");
session.setAttribute("password", "admin");
```

To get the value from session we use the **getAttribute() method of HttpSession interface**. Here we are fetching the attribute values using attribute names.

```
String userName = (String) session.getAttribute("username");
```

Methods of HttpSession

- **public void setAttribute(String name, Object value)**: Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.
- **public Object getAttribute(String name)**: Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.
- **public Enumeration getAttributeNames()**: Returns an Enumeration that contains the name of all the objects that are bound as attributes to the session object.
- **public void removeAttribute(String name)**: Removes the given attribute from session.
- **setMaxInactiveInterval(int interval)**: Sets the session inactivity time in seconds. This is the time in seconds that specifies how long a sessions remains active since last request received from client.

Example of HttpSession

index.html

```
<html>
<body>
     <a href="first">Set Session</a>
     <a href="second">Access Session</a>
</body>
</html>
```

FirstServlet.java

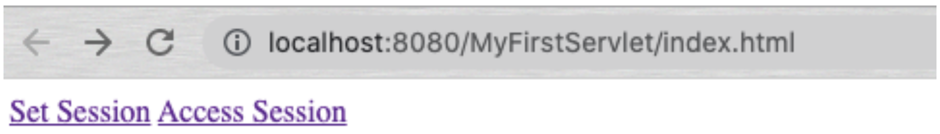
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.servlet.annotation.*;

@WebServlet("/first")
public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException,IOException{

           //setting a session
           HttpSession session=req.getSession();
           session.setAttribute("userid", "10115");
           session.setAttribute("username", "Raaju");

           res.setContentType("text/html");
           PrintWriter out=res.getWriter();
           out.println("Session Set Successfully!");
     }
}
```

SecondServlet.java

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

@WebServlet("/second")
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException,IOException{

           //accessing session
        HttpSession session=req.getSession(false);
        String userId=(String) session.getAttribute("userid");
        String username=(String) session.getAttribute("username");
```

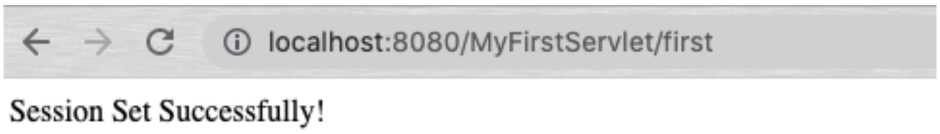
```
            //displaying session values
            res.setContentType("text/html");
            PrintWriter out=res.getWriter();

            out.println("User Id: "+userId+"<br>");
            out.println("Username: "+username);

        }
}
```

Output:



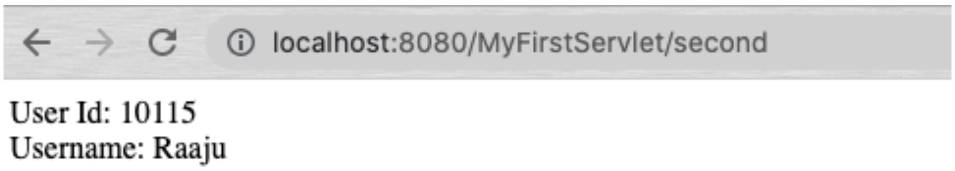
```
localhost:8080/MyFirstServlet/index.html
```

Set Session Access Session



```
localhost:8080/MyFirstServlet/first
```

Session Set Successfully!



```
localhost:8080/MyFirstServlet/second
```

User Id: 10115
Username: Raaju

CRUD operation using Servlet

- As we know that for database operation, we need JDBC driver JAR file. Since, we create web application using servlet, **we need to put JAR file under webapp/WEB-INF/lib directory.**

DbConnection.java

```
import java.sql.*;
public class DbConnection {
    public static Connection getConn() throws Exception{
      Class.forName("org.sqlite.JDBC");
      String dbUrl="jdbc:sqlite:mydb";
      Connection conn=DriverManager.getConnection(dbUrl);

      Statement st=conn.createStatement();
      String sql="CREATE TABLE If not exists employees(eid INT,
           name VARCHAR(30), address VARCHAR(30))";
      st.execute(sql);
      return conn;
    }
}
```

Index.html

```
<html>
<head>
<title>Employee Crud</title>
</head>
<body>
    <form method="POST" action="myservlet">
    Employee Id: <input type="text" name="eid"/>
    <br>
    Name: <input type="text" name="name"/>
    <br>
    Address: <input type="text" name="address"/>
    <br> <br>
    <input type="submit" value="Insert" name="insert"/>
    <input type="submit" value="Update" name="update"/>
    <input type="submit" value="Delete" name="delete"/>
    </form>
    <br><br>
    <a href="view">View Employee Records</a>
</body>
</html>
```

MyServlet.java

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

@WebServlet("/myservlet")
public class MyServlet extends HttpServlet{

public void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException,IOException{

        //getting request values
        int eid=Integer.parseInt(req.getParameter("eid"));
        String name=req.getParameter("name");
        String address=req.getParameter("address");

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

           try {
              Connection conn=DbConnection.getConn();

              //handling button clicks
              if(req.getParameter("insert")!=null) {
                    //insert button clicked
                    //inserting data
                    String sql="INSERT INTO employees
                          (eid,name,address) VALUES (?,?,?)";
                    PreparedStatement pst=conn.prepareStatement(sql);
                    pst.setInt(1,eid);
                    pst.setString(2, name);
                    pst.setString(3, address);
                    pst.executeUpdate();
                    //displaying message in javascript alert
              out.println("<script>alert('Inserted Successfully!');"
                         +"window.location.href='view'</script>");
              }

              else if(req.getParameter("update")!=null) {
                    //update button Clicked
                    String sql="UPDATE employees SET name=?,
                       address=? WHERE eid=?";
                    PreparedStatement pst=conn.prepareStatement(sql);
                    pst.setString(1, name);
                    pst.setString(2, address);
                    pst.setInt(3,eid);
                    pst.executeUpdate();
                    //displaying message in javascript alert
              out.println("<script>alert('Updated Successfully!');"
                         +"window.location.href='view'</script>");
              }
```

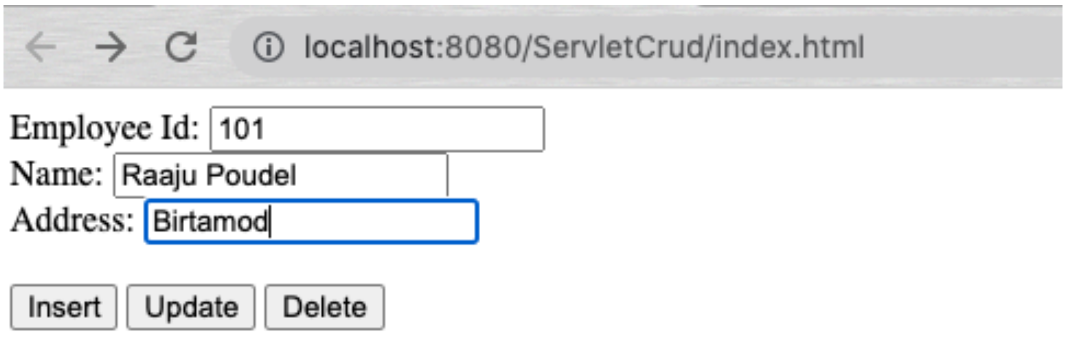
```
          else if(req.getParameter("delete")!=null) {
                //delete button clicked
                String sql="DELETE FROM employees WHERE eid=?";
                PreparedStatement pst=conn.prepareStatement(sql);
                pst.setInt(1, eid);
                pst.executeUpdate();
                //displaying message in javascript alert
          out.println("<script>alert('Deleted Successfully!');"
                     +"window.location.href='view'</script>");
           }

        }catch(Exception ex) {
           System.out.println(ex.toString());
        }

    }
}
```

ViewServlet.java

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

@WebServlet("/view")
public class ViewServlet extends HttpServlet{

public void doGet(HttpServletRequest req, HttpServletResponse res)
         throws ServletException,IOException{

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

       //selecting data
          try {
            Connection conn=DbConnection.getConn();
            String sql="SELECT * FROM employees";
            PreparedStatement pst=conn.prepareStatement(sql);
            ResultSet rs=pst.executeQuery();

            out.println("<html><body>");
            out.println("<a href='index.html'>
                          Goto Index </a> <br><br>");
            //displaying data
            out.println("<table>");
            out.println("<tr>");
            out.println("<th> Eid </th>");
            out.println("<th> Name </th>");
            out.println("<th> Address </th>");
            out.println("</tr>");
```

```
                while(rs.next()) {
                   out.println("<tr>");
                   out.println("<td>"+rs.getInt(1)+"</td>");
                   out.println("<td>"+rs.getString(2)+"</td>");
                   out.println("<td>"+rs.getString(3)+"</td>");
                   out.println("<td>");
                   out.println("</td>");
                   out.println("</tr>");
               }

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


            }catch(Exception ex) {
                System.out.println(ex.toString());
            }
        }
}
```

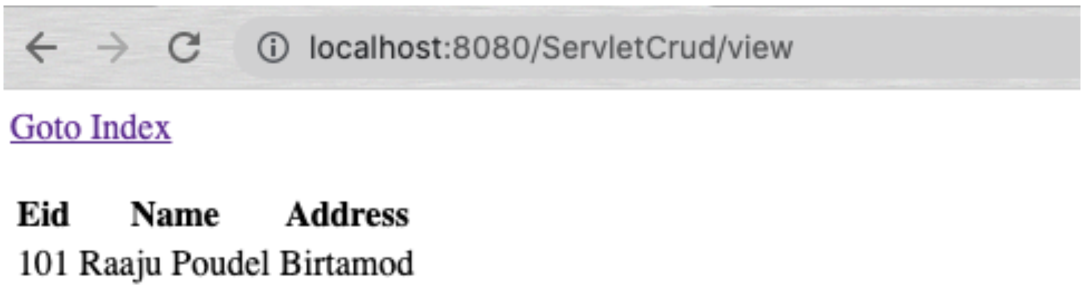
Output:



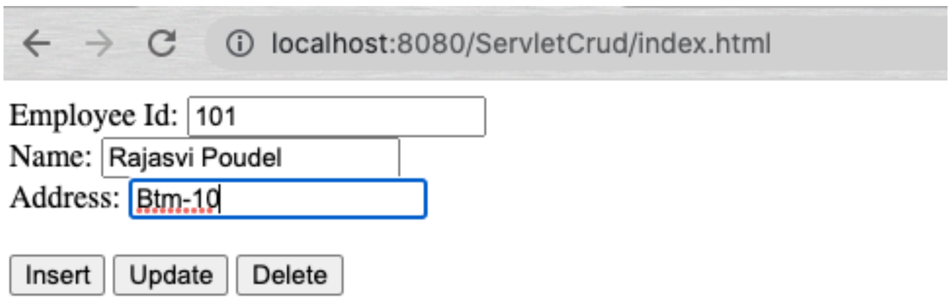
```
localhost:8080/ServletCrud/index.html

Employee Id: 101
Name: Raaju Poudel
Address: Birtamod

Insert  Update  Delete
```

View Employee Records

After Insert,

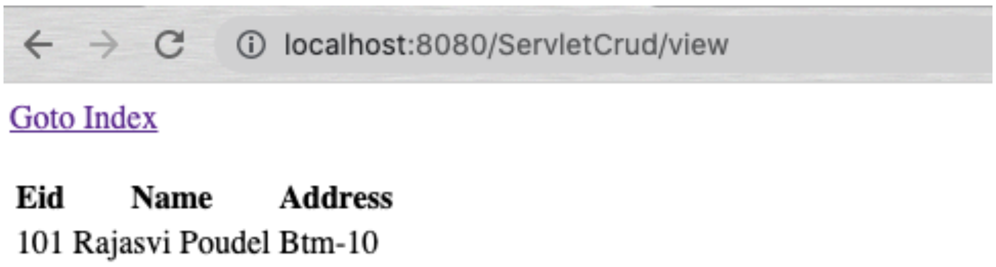


```
localhost:8080/ServletCrud/view

Goto Index

 Eid    Name     Address
 101 Raaju Poudel Birtamod
```


localhost:8080/ServletCrud/index.html

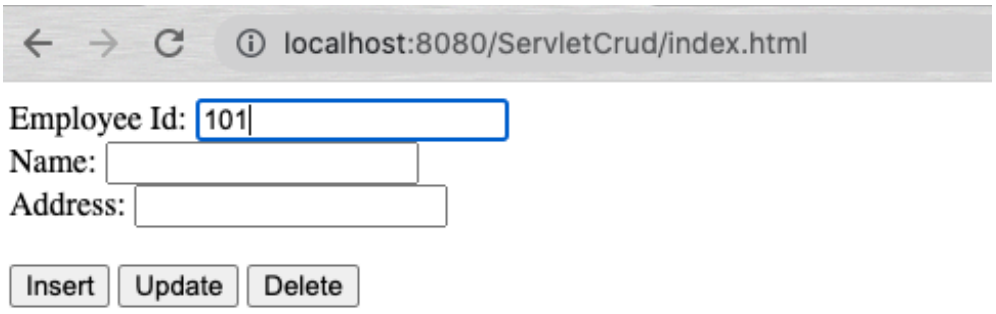
Employee Id: 101
Name: Rajasvi Poudel
Address: Btm-10

After Update,


localhost:8080/ServletCrud/view

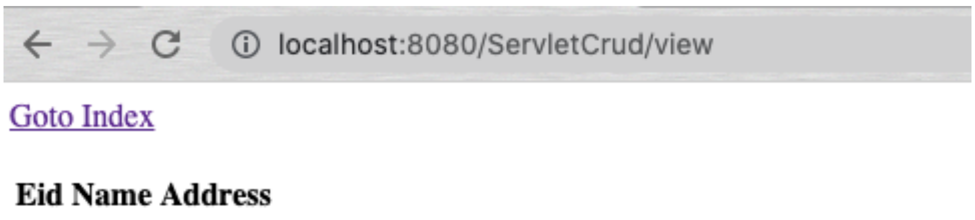
Goto Index

```
Eid   Name   Address
101 Rajasvi Poudel Btm-10
```


localhost:8080/ServletCrud/index.html

Employee Id: 101
Name:
Address:

After Delete,


localhost:8080/ServletCrud/view

Goto Index

```
Eid Name Address
```

Java Server Page (JSP)

JSP is a server side technology that does all the processing at server. It is used for creating dynamic web applications, using java as programming language. **It can be thought of as an extension to Servlet** because it provides more functionality than servlet.
A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Basically, any html file can be converted to JSP file by just changing the file extension from ".html" to ".jsp", it would run just fine. What differentiates JSP from HTML is the ability to use java code inside HTML. In JSP, you can embed Java code in HTML using JSP tags.

```
<%-- JSP comment --%>
<HTML>
<HEAD>
<TITLE>MESSAGE</TITLE>
</HEAD>
<BODY>
<%out.print("Hello, My First JSP code");%>
</BODY>
</HTML>
```

Advantages of JSP

1. JSP has all the advantages of servlet, like: Better performance than CGI Built in session features, it also inherits the the features of java technology like – multithreading, exception handling, Database connectivity,etc.
2. JSP Enables the separation of content generation from content presentation. Which makes it more flexible.
3. With the JSP, it is now easy for web designers to show case the information what is needed.
4. Web Application Programmers can concentrate on how to process/build the information.

Servlet Vs JSP

Like JSP, Servlets are also used for generating dynamic webpages. Here is the comparison between them.
The major difference between them is that servlet adds HTML code inside java while JSP adds java code inside HTML. There are few other noticeable points that are as follows:

Servlets –

1. Servlet is a Java program which supports HTML tags too.
2. Generally used for developing business layer(the complex computational code) of an enterprise application.
3. Servlets are created and maintained by Java developers.

JSP –

1. JSP program is a HTML code which supports java statements too.To be more precise, JSP embed java in html using JSP tags.
2. Used for developing presentation layer of an enterprise application
3. Frequently used for designing websites and used by web developers.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

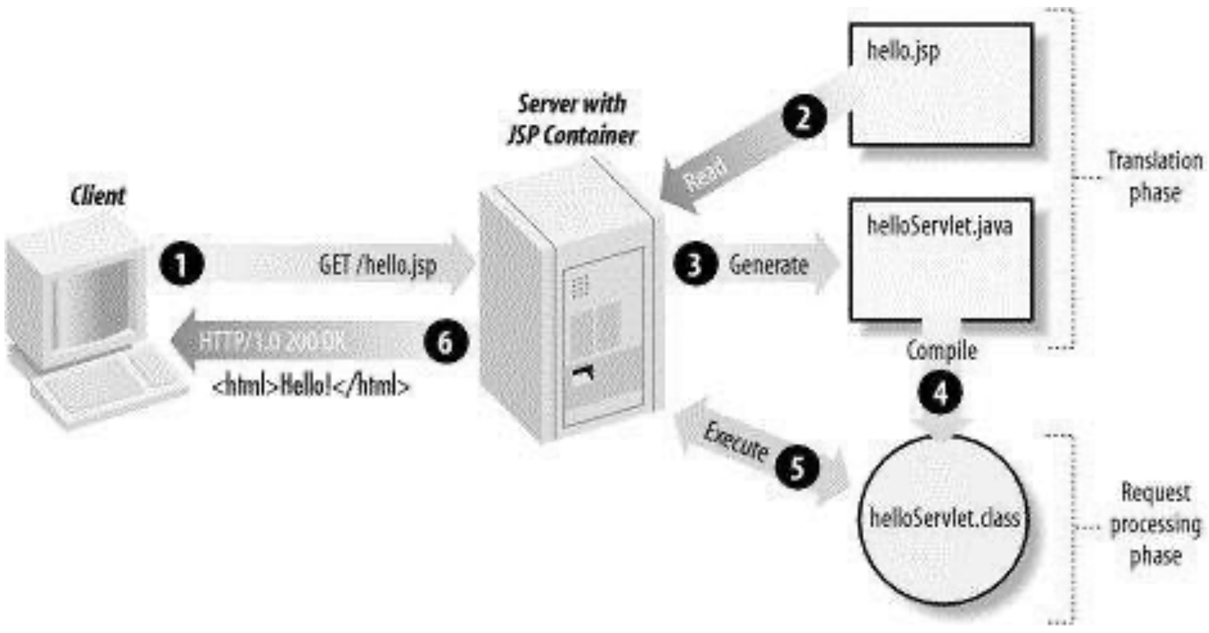
3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

JSP Architecture



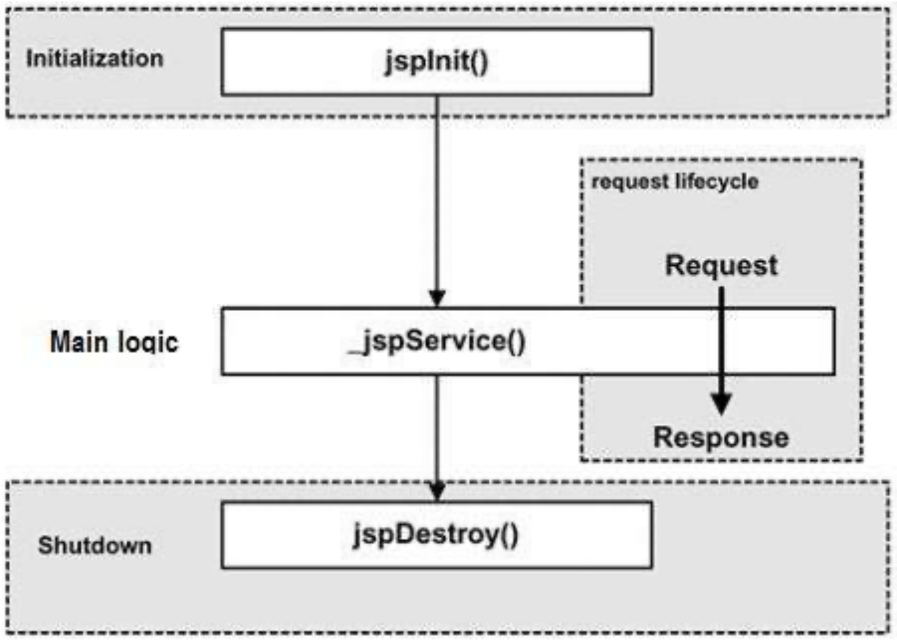
The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

Lifecycle of JSP

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases are –

- Compilation
- Initialization
- Execution
- Cleanup



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
The compilation process involves three steps −

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method −

```
public void jspInit(){
   // Initialization code...
}
```

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.
Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.
The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −

```
void _jspService(HttpServletRequest request, HttpServletResponse
response) {
   // Service handling code...
}
```

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

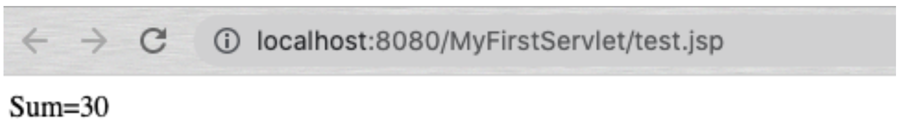
The jspDestroy() method has the following form −

```
public void jspDestroy() {
   // Your cleanup code goes here.
}
```

Steps for creating and Running a JSP in Tomcat Server using Eclipse IDE:

1. Create a Dynamic Web Project.
2. Create a JSP page under webapp directory.
3. Write following JSP code:
```
<html>
<head>
<title>Insert title here</title>
</head>
<body>
    <%
    int a=10,b=20,sum;
    sum=a+b;
    out.print("Sum="+sum);
    %>
</body>
</html>
```
4. Run the project.

Output:


localhost:8080/MyFirstServlet/test.jsp

```
Sum=30
```

JSP Syntax

There are **five main tags** that all together form JSP syntax. These tags are: **Declaration tags, Expression tags, Directive tags, Scriptlet tags and Action tags.**

Declaration Tag (<%!   %>)

This tag allows the developer to declare variables or methods. This doesn't generate output so are used with JSP expressions or scriptlets tags.

Example
```
<%!
        private int id=101;
        public void getSalary(int id);
%>
```

Directive Tag (<%@ directive attribute="value" %>)

This tag gives special information about the page to the JSP engine. There are three main types of directives:

- Page Directive – processing information for this page.
- Include Directive – files to be included.
- Tag library Directive – tab library to be used in this page.

Page Directive:

This directive has 11 optional attributes that provide the JSP engine with special processing information. Some of the attributes are:

```
<%@ page language = ”java”  %>
<%@ page extends = ”com.example….” %>
<%@ page import = ”java.util.*”  %>
```

Include Directive:

It allows developer to include contents of a file inside another. File is included during translation phase.

```
<%@ include file = “index.html” %>
<%@ include file = “test.jsp” %>
```

Tab Library Directive:

A tag lib is a collection of custom tags that can be used by web developers.

```
<%@ tablib uri = “http://www.raaju.com/tags” prefix=“mytag” %>
```

Scriptlet Tag (<% …. %>)

In JSP, java code can be written inside the jsp page using the scriptlet tag.

```
<%
     String name = “Raju Poudel”
     out.println(name);
%>
```

Action Tag:

The action tags are used to control the flow between pages and to use Java Bean. Some of the action tags supported by JSP are:

- jsp:forward
- jsp:include
- jsp:param

Following example forwards action to another JSP page:

```
<jsp: forward page = “nextpage.jsp” />
```

Comments in JSP

Following is the syntax of JSP comments:

```
<%-- This is a single line comment   --%>

<%--
     This is a
      multi-line comment
--%>
```

JSP Implicit Objects

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within **Scriptlet** without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Objects and their corresponding classes:

out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

1. **Out**: This is used for writing content to the client (browser). It has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer.

2. **Request**: The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purposes.

3. **Response**: It is basically used for modfying or delaing with the response which is being sent to the client(browser) after processing the request.

4. **Session:** It is most frequently used implicit object, which is used for storing the user's data to make it available on other JSP pages till the user session is active.

5. **Application:** This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.

6. **Exception:** Exception implicit object is used in exception handling for displaying the error messages. This object is only available to the JSP pages, which has isErrorPage set to true.

7. **Page:** Page implicit object is a reference to the current Servlet instance (Converted Servlet, generated during translation phase from a JSP page). We can simply use **this** in place of it.

8. **pageContext**: It is used for accessing page, request, application and session attributes.

9. **Config:** This is a Servlet configuration object and mainly used for accessing getting configuration information such as servlet context, servlet name, configuration parameters etc.

Handling form data using JSP

Form handling in JSP is similar to servlets. We can use request object for reading parameter values from HTML file or other JSP file. Methods supported by request object is similar to methods used in servlets.

index.html

```
<html>
<body>
     <form method="POST" action="add.jsp">
         First Number: <input type="text" name="first"/>
         <br>
         Second Number: <input type="text" name="second"/>
         <br><br>
         <input type="submit" value="Calculate"/>
     </form>
</body>
</html>
```

add.jsp

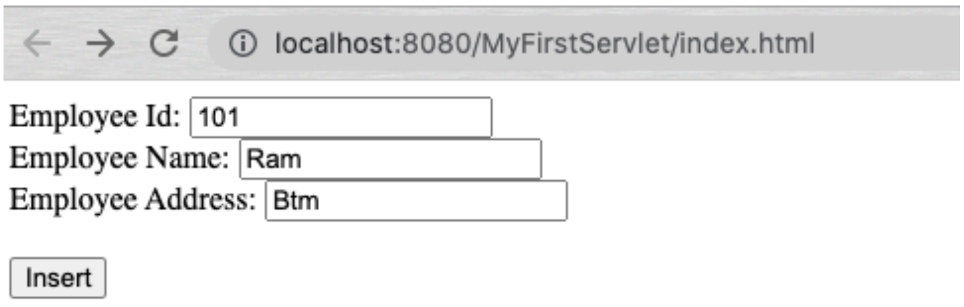
```
<html>
<body>
     <%!
        private int a,b,sum;
     %>

     <%
       a=Integer.parseInt(request.getParameter("first"));
       b=Integer.parseInt(request.getParameter("second"));
       sum=a+b;
      %>

    <h2 style="color:blue;">
       <%  out.println("Sum="+sum); %>
    </h2>
</body>
</html>
```

Working with database using JSP:

index.html
```
<html>
<body>
     <form method="POST" action="insert.jsp">
         Employee Id: <input type="text" name="eid"/>
         <br>
         Employee Name: <input type="text" name="name"/>
         <br>
         Employee Address: <input type="text" name="address"/>
         <br><br>
         <input type="submit" value="Insert"/>
     </form>
</body>
</html>
```

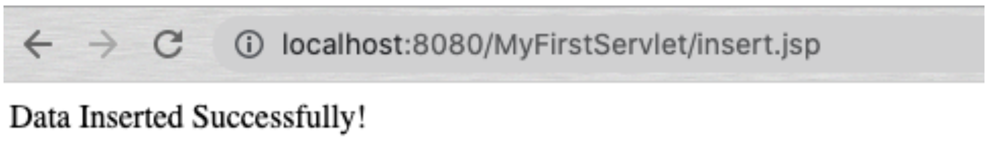
insert.jsp
```
<html>
<body>
    <%
       int eid=Integer.parseInt(request.getParameter("eid"));
       String name=request.getParameter("name");
       String address=request.getParameter("address");
    %>

    <%@ page import="java.sql.*" %>

    <%
       Class.forName("org.sqlite.JDBC");
       String dbUrl="jdbc:sqlite:mydb";
        Connection conn=DriverManager.getConnection(dbUrl);

        Statement st=conn.createStatement();
        String sql="CREATE TABLE If not exists employee(eid INT,
                name VARCHAR(30), address VARCHAR(30))";
        st.execute(sql);

        //inserting data
        String sql1="insert into employee(eid,name,address) values
                         (?,?,?)";
        PreparedStatement pst=conn.prepareStatement(sql1);
        pst.setInt(1, eid);
        pst.setString(2,name);
        pst.setString(3,address);
        pst.executeUpdate();

        out.println("Data Inserted Successfully!");
    %>

</body>
</html>
```

Output:


localhost:8080/MyFirstServlet/index.html

Employee Id: 101
Employee Name: Ram
Employee Address: Btm

Insert


localhost:8080/MyFirstServlet/insert.jsp

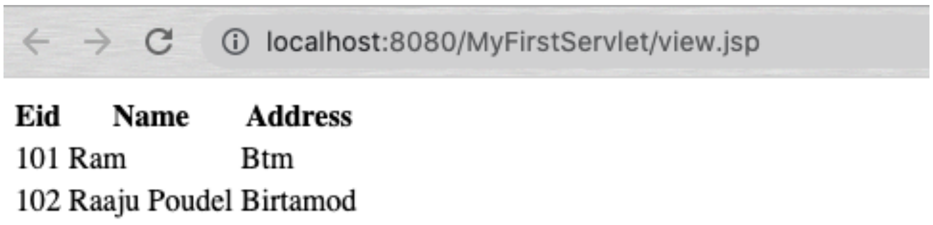
Data Inserted Successfully!

Selecting and displaying data in Table:

view.jsp

```
<html>
<body>
   <%@ page import="java.sql.*" %>
       <%
          Class.forName("org.sqlite.JDBC");
          String dbUrl="jdbc:sqlite:mydb";
          Connection conn=DriverManager.getConnection(dbUrl);

           //selecting data
           String sql="select * from employee";
           PreparedStatement pst=conn.prepareStatement(sql);
           ResultSet rs=pst.executeQuery();
        %>
    <table>
       <tr>
            <th> Eid </th>
            <th> Name </th>
            <th> Address </th>
        </tr>
         <%
               while(rs.next()){
                 out.print("<tr>");
                   out.print("<td>"+rs.getInt(1)+"</td>");
                   out.print("<td>"+rs.getString(2)+"</td>");
                   out.print("<td>"+rs.getString(3)+"</td>");
                   out.print("</tr>");
               }
         %>
      </table>
</body>
</html>
```

Output


localhost:8080/MyFirstServlet/view.jsp

```
Eid   Name        Address
101 Ram           Btm
102 Raaju Poudel Birtamod
```

Session Management in JSP

Setting a session in JSP,

```
session.setAttribute("key",value);
```

Accessing a session in JSP,

```
session.getAttribute("key");
```

Example is shown below:

index.html

```
<html>
<body>
    <form method="POST" action="setsession.jsp">
        Username:
        <input type="text" name="username"/>
        <br>
        <input type="submit" value="Submit"/>
    </form>
</body>
</html>
```

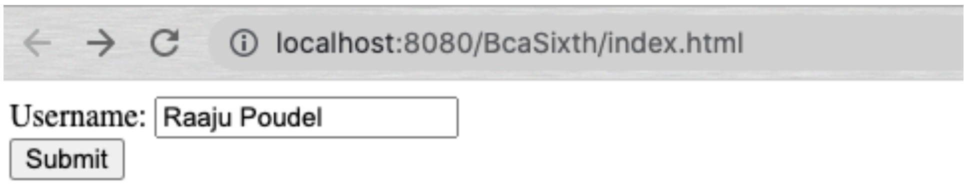
setsession.jsp

```
<html>
<body>
    <%
       String uname=request.getParameter("username");
       session.setAttribute("username", uname);
       out.println("Session set Successfully!");
    %>

    <a href="viewsession.jsp">Click to access session</a>
</body>
</html>
```

viewsession.jsp

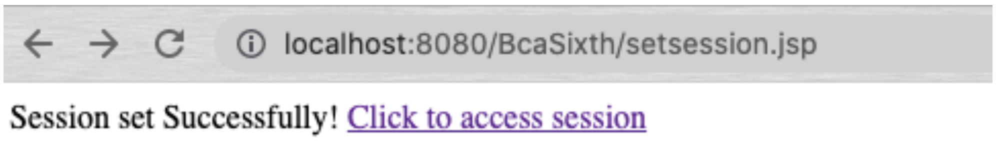
```
<html>
<body>
     <%
        String uname=(String)session.getAttribute("username");
        out.println("Welcome: "+uname);
     %>
</body>
</html>
```

Output:

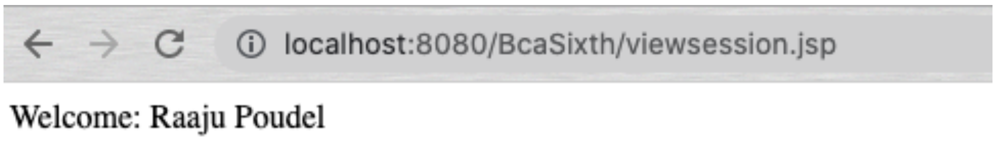

localhost:8080/BcaSixth/index.html

Username: Raaju Poudel

Submit


localhost:8080/BcaSixth/setsession.jsp

Session set Successfully! Click to access session


localhost:8080/BcaSixth/viewsession.jsp

Welcome: Raaju Poudel

Exception Handling in JSP

In JSP, **exception is an implicit object of type java.lang.Throwable** class. This object can be used to print the exception. But it can only be used in error pages.
The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Exception handling by the elements of page directive

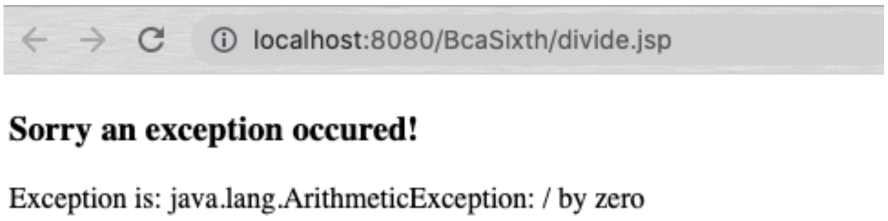
error.jsp

```
<html>
<body>
     <%@ page isErrorPage="true" %>

     <h3>Sorry an exception occured!</h3>

     Exception is: <%= exception %>

</body>
</html>
```

divide.jsp

```
<html>
<body>
    <%@page errorPage="error.jsp" %>
    <%
        int a=10,b=0,div;
        div=a/b;
        out.println("Result="+div);
    %>
</body>
</html>
```

Output:


localhost:8080/BcaSixth/divide.jsp

Sorry an exception occured!

Exception is: java.lang.ArithmeticException: / by zero

Exception handling by specifying the error-page element in web.xml file

web.xml

```
<error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/error.jsp</location>
</error-page>
```

divide.jsp

Now, you don't need to specify the errorPage attribute of page directive in the jsp page.

```
<html>
<body>
    <%
        int a=10,b=0,div;
        div=a/b;
        out.println("Result="+div);
    %>
</body>
</html>
```

error.jsp file is same as in the above example

Output is also same as in the above example

Java Web Frameworks:

Frameworks are tools with pre-written code, act as a template or skeleton, which can be reused to create an application by simply filling with your code as needed which enables developers to program their application with no overhead of creating each line of code again and again from scratch. Now if you are searching for the best tools in the market that help you to develop a website easier, faster and better then Java is always there for you with a large number of frameworks and most importantly most of the Java frameworks are free and open-source.

1. Spring:

Spring is a powerful, lightweight, and most popular framework which makes Java quicker, easier, and safer to use. This framework is very popular among developers for its speed, simplicity, and productivity which helps to create enterprise-level web applications with complete ease. Spring MVC and Spring Boot made Java modern, reactive, and cloud-ready to build high-performance complex web applications hence used by many tech-giants, including Netflix, Amazon, Google, Microsoft, etc.

- Using Spring's flexible and comprehensive third-party libraries you can build any web application you can imagine.
- Start a new Spring project in seconds, with, fast startup, fast shutdown, and optimized execution by default.
- Spring provides a lightweight container that can be triggered without a web server or application server.
- It provides backward compatibility and easy testability of your project.
- It supports JDBC which improves productivity and reduces the error as much as possible
- It supports modularity and both XML and annotation-based configuration
- Spring Boot has a huge ecosystem and community with extensive documentation and multiple Spring tutorials.

2. Grails:

It is a dynamic full-stack Java framework based on the MVC design pattern. which is easy to learn and most suitable for beginners. Grails is an object-oriented language that enhances developer productivity. While written in Groovy it can run on the Java platform and is perfectly compatible with Java syntax.

- Easy to Creating tags for the View,
- built-in support for RESTful APIs,
- you can mix Groovy and Java using Grails,
- best suitable for Rapid Development,
- configuration features are dynamic, no need to restart the server.

3. Google Web Toolkit (GWT):

It is a very popular open-source Java framework used by a large number of developers around the world for building and optimizing complex browser-based applications. This framework is used for the productive development of high-performance complex web applications without being an expert in front-end technologies like JavaScript or responsive design. It converts Java code into JavaScript code which is a remarkable feature of GWT. Popular Google's applications like AdSense and AdWords are written and using this framework.

- Google APIs are vastly used in GWT applications.
- Open-source and Developer-friendly.
- Easily create beautiful UIs without vast knowledge in front-end scripting languages.
- Create optimized web applications that are easy to debug.
- Compiles the Java source code into JavaScript files that can run on all major browsers.

4. Struts

It is a very useful framework, an open-source MVC framework for creating modern enterprise-level Java web applications that favor convention over configuration and reduce overall development time. It comes with plugins to support REST, AJAX, and JSON and can be easily integrated with other Java frameworks like Spring and Hibernate.

- Super Flexible and beginner-friendly,
- Reliable, based on MVC design pattern.
- Integration with REST, JSON, and AJAX.
- Creative themes and templates make development tasks faster.
- Extend capabilities for complex development,
- Reduced development time and Effort, makes dev easier and fun.

5. JavaServer Faces (JSF)

It is quite similar to Struts, a free web application developed framework, maintained by Oracle technology which simplifies building user interfaces for Server-side applications by assembling reusable UI components in a page. JSF is a component-based MVC framework that encapsulates various client-side technologies and focuses more on the presentation layer to allow web developers to create UI simply by just drag and drop.

- Rich libraries and reusable UI components,
- Easy front end tools to use without too much coding,
- Jsf helps to improve productivity and consistency,
- Enrich the user experience by adding Ajax events for validations and method invocations.
- It provides an API to represent and manage UI components and instead of using Java, JSF uses XML for view handling.

6. Hibernate

A stable, lightweight ORM Java framework that can easily communicate with any database and is more convenient when working with multiple databases. Working with Hibernate is fun using powerful APIs and several useful tools like Mapping Editor, Wizards, and Reverse Engineering. Many big companies including Platform, DAILY HOTEL, IBM, and Dell use Hibernate in their tech stacks,

- Light-weight and easy to scale up, modify, and configure.
- Complex data manipulation with less coding.
- High productivity and portability,
- Used for RDBMS as well as the NoSQL database.
- Awesome Command-line tools and IDE plugins to makes your experience pleasant.

7. Play

A unique type of framework that makes easier to build a web application using Java and follows the approach of convention over configuration. It is based on the stateless, web-friendly, and lightweight architecture, the MVC pattern. It provides minimal resource consumption (CPU & memory) for a highly scalable modern mobile and web application.

- High performance due to asynchronous processing
- Reactive principles improve the productivity of developer,
- Most of the errors are caught during compile time saving a lot of mistakes early in the development life-cycle.
- Easy and quick reload for any changes in the configuration,
- Easy to create simple JAR files.

8. Vaadin

An open-source client-server framework, with an active worldwide community that allows you to create complex and dynamic web applications using pre-designed UI components. Write UI in plain Java without even bothering JS, HTML, and CSS. You can also create layouts with a visual designer instead of HTML. Vaadin provides a server-side architecture which helps developers to create dynamic and interactive interfaces for the web. Using Vaadin access to the DOM directly from the Java virtual machine. The updated components can be combined with other frontend JavaScript technologies such as React and Vue, or even plain JavaScript.

- High developer productivity and fast development;
- Built on the Web Components standards,
- Provides many components, and different listeners,
- Automates client-server communication and routing,
- Has good documentation and an active community.

9. Wicket

Wicket is a very simple Java web framework, with a component-oriented structure where all you need to know, is only Java and HTML with no XMLs or configuration files. Previous experience of working with JSP makes working with Wicket, simply a cakewalk. The main feature of Wicket is its POJO model wherein components are simple (Plain Old) Java Objects having OOP features. These components come in bundled together as reusable packages so that developers can customize them with images, buttons, forms, links, pages, containers, behaviors, and more.

- light-weight and superfast framework.
- Unit testing is very easy with Wicket.
- Zero XML configuration files,
- No Back-Button problem,
- Easy to create Bookmarkable Projects.

10. Dropwizard

A light-weight Java framework with out of the box support for advanced configurations, lets you complete your application in the fastest way. It is a magical framework with brilliantly integrated libraries like Jetty, Guava, Jersey, Jackson, and Metrics for all the configurations, security, and performance-related tasks. The developer just needs to work on building your business logic without any extra overhead. Any beginner can create high-performance RESTful web applications very easily using Dropwizard.

- Easy to perform rapid prototyping,
- Support open sources and independent libraries,
- Quick Project Bootstrap,
- Easy to set up and beginner-friendly
- Build high performance, stable, and reliable web application.

So If you want to develop an amazing and reliable web application then you can use any of these top Java frameworks as these will meet your business requirements and will offer a certain level of flexibility with optimum performance and security. Using the right frameworks makes web development fun So Choose your framework wisely to enjoy all the features that Java provides.