**Servlets**

**1.. Introduction**

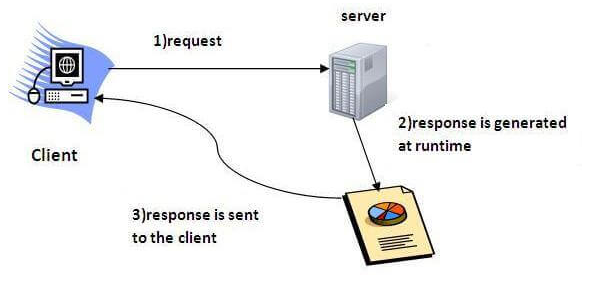
Servlet is a Simple Java Program that runs on the server and it is capable of handling requests, processing the requests and creates a dynamic response. Hence, Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

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

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

So, in brief, 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.



**Note**: We should download, install and configure any of the Web Server in our machine and in our IDE in order to run any Servlet related Program. The Recommended Web Server is Apache Tomcat Web Server.

**Note**: Changing the port number is required if there is another server running on the same system with same port number. Suppose you have installed oracle, you need to change the port number of Apache tomcat because both have the default port number 8080. Open server.xml file in notepad. It is located inside the apache-tomcat/conf directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

**2.. 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.

**2.1 Interfaces in javax.servlet package**

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

* Servlet
* ServletRequest
* ServletResponse
* RequestDispatcher
* ServletConfig
* ServletContext
* SingleThreadModel
* Filter
* FilterConfig
* FilterChain
* ServletRequestListener
* ServletRequestAttributeListener
* ServletContextListener
* ServletContextAttributeListener

**2.2 Classes in javax.servlet package**

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

* GenericServlet
* ServletInputStream
* ServletOutputStream
* ServletRequestWrapper
* ServletResponseWrapper
* ServletRequestEvent
* ServletContextEvent
* ServletRequestAttributeEvent
* ServletContextAttributeEvent
* ServletException
* UnavailableException

**2.3 Interfaces in javax.servlet.http package**

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

* HttpServletRequest
* HttpServletResponse
* HttpSession
* HttpSessionListener
* HttpSessionAttributeListener
* HttpSessionBindingListener
* HttpSessionActivationListener
* HttpSessionContext (deprecated now)

**2.4 Classes in javax.servlet.http package**

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

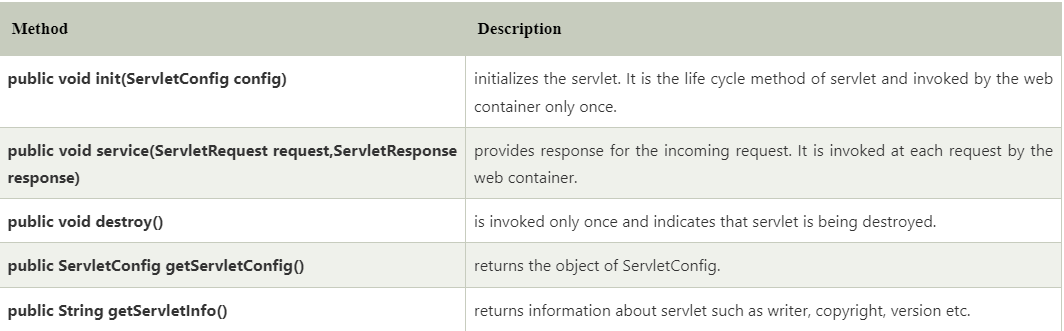
* HttpServlet
* Cookie
* HttpServletRequestWrapper
* HttpServletResponseWrapper
* HttpSessionEvent
* HttpSessionBindingEvent
* HttpUtils (deprecated now)

**3.. Creating Servlet by Implementing Servlet Interface**

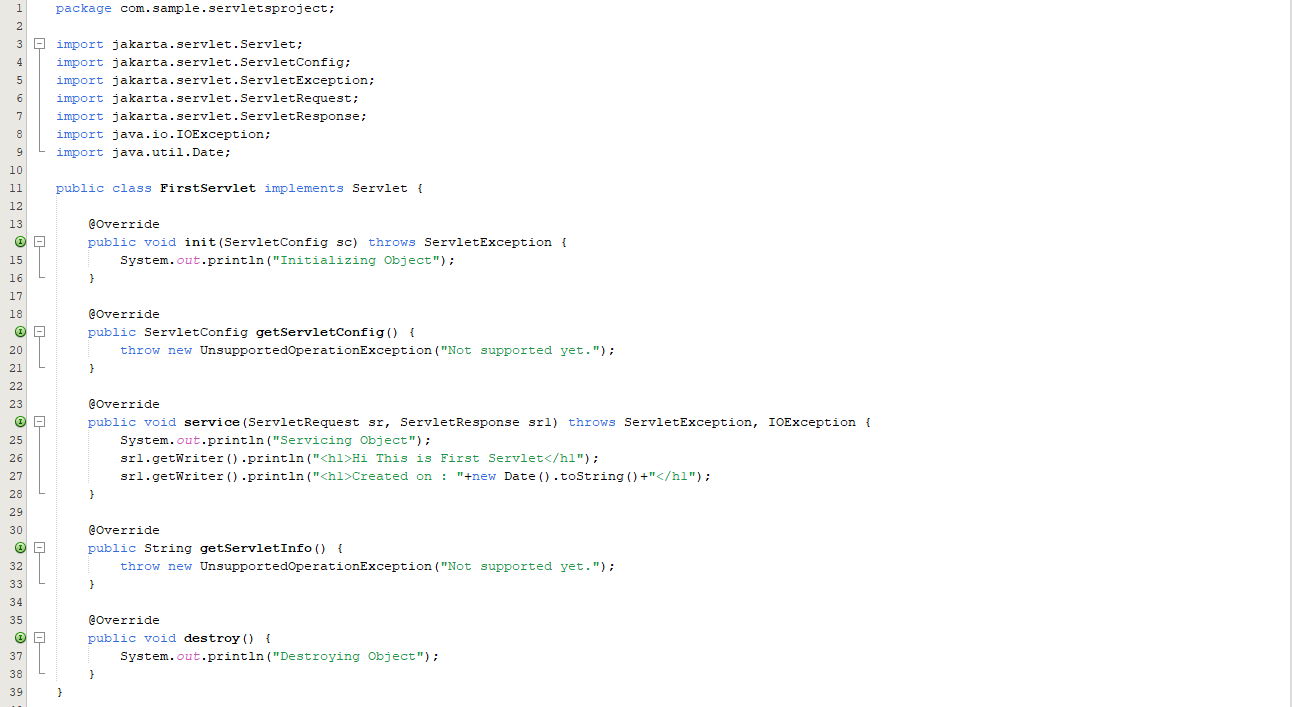
Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet,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

**3.1 Methods of Servlet interface**

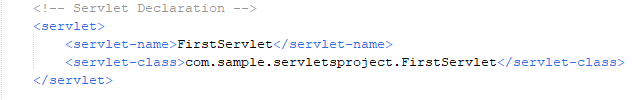
There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.



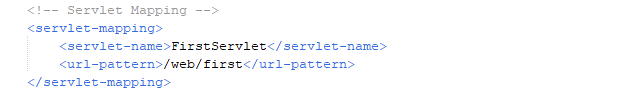
To Create any Custom Servlet, we have to create a new class and this new class should implement the methods present in Servlet Interface.



After creating class we are required to create Servlet in web.xml file. This is how we will declare any Servlet in web.xml file.

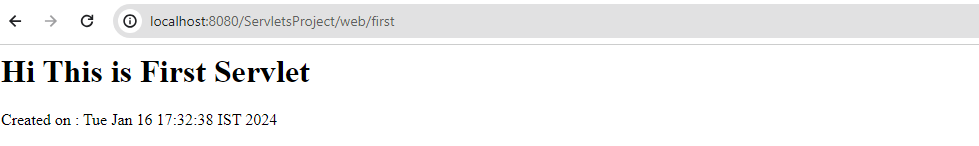


Here, servlet-name can be any name by which we wants to defined our Servlet and servlet-class is a fully qualified class name. Once we declare a Servlet we have define Servlet mapping for the created Servlet. Servlet mapping means by hitting which URL our servlet will be called. This is how we create Servlet Mapping in web.xml file.



Servlet-name in Servlet declaration and in servlet-mapping should be same.

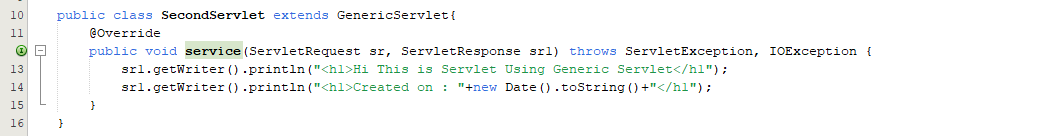
Now when we run the application and hit the given URL, we will get the output which we have specified in service() method of our servlet class.



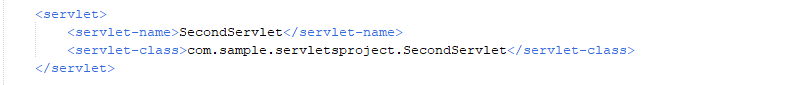
**Note**: In September 2017, all interfaces and classes related to Servlet are part of Jakarta instead of javafx. In September 2017, Oracle decided to give away the rights for Java EE to the Eclipse Foundation (the language is still owned by Oracle). Actually, the Eclipse Foundation legally had to rename Java EE. That’s because Oracle has the rights over the “Java” brand. So, to choose the new name, the community voted and picked: **Jakarta** **EE**. In a certain way, it’s still JEE.

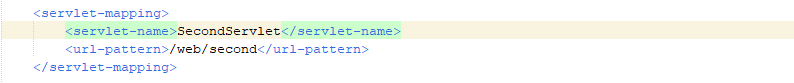
**4.. Creating Servlet by Extending GenericServlet Class**

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method which is declared as an abstract method in class and hence, GenericServlet is an Abstract class. GenericServlet class can handle any type of request so it is protocol-independent. We can create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.



Declaration and Mapping of this Servlet will be same as previous One. In web.xml file we will declare Servlet and Mapping as Below:





Let’s run it and check Output.



**4.1 Methods of GenericServlet class**

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

* **public void init(ServletConfig config)**

It is used to initialize the servlet.

* **public abstract void service(ServletRequest request, ServletResponse response)**

It provides service for the incoming request. It is invoked at each time when user requests for a servlet.

* **public void destroy()**

It is invoked only once throughout the life cycle and indicates that servlet is being destroyed.

* **public ServletConfig getServletConfig()**

It returns the object of ServletConfig.

* **public String getServletInfo()**

It returns information about servlet such as writer, copyright, version etc.

* **public void init()**

It is a convenient method for the servlet programmers, now there is no need to call super.init(config)

* **public ServletContext getServletContext()**

It returns the object of ServletContext.

* **public String getInitParameter(String name)**

It returns the parameter value for the given parameter name.

* **public Enumeration getInitParameterNames()**

It returns all the parameters defined in the web.xml file.

* **public String getServletName()**

It returns the name of the servlet object.

* **public void log(String msg)**

It writes the given message in the servlet log file.

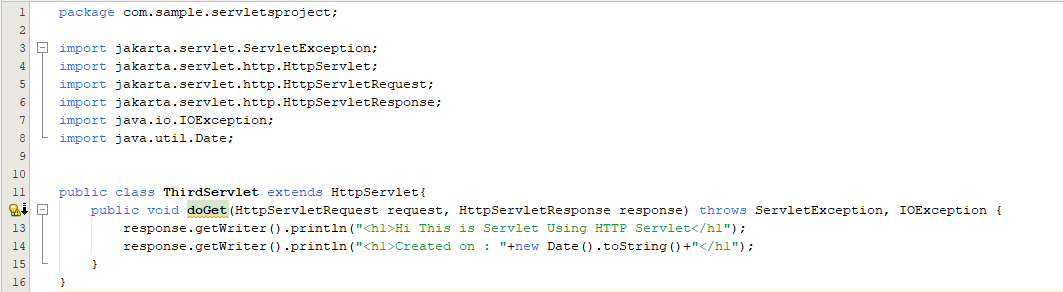
* **public void log(String msg,Throwable t)**

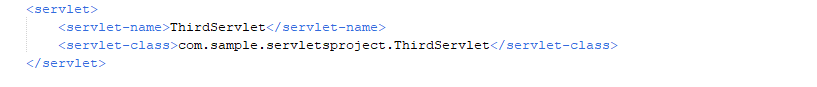
It writes the explanatory message in the servlet log file and a stack trace.

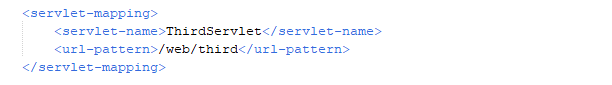
**5.. Creating Servlet by Extending HttpServlet Class**

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

Let’s implement doGet method of HttpServlet class.

Mapping in web.xml will be same as before.





Let’s run the application and hit the endpoint.



**5.1 Methods of HttpServlet class**

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

* **public void service(ServletRequest req, ServletResponse res)**

Dispatches the request to the protected service method by converting the request and response object into http type.

* **protected void service(HttpServletRequest req, HttpServletResponse res)**

It receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.

* **protected void doGet(HttpServletRequest req, HttpServletResponse res)**

It handles the GET request. It is invoked by the web container.

* **protected void doPost(HttpServletRequest req, HttpServletResponse res)**

It handles the POST request. It is invoked by the web container.

* **protected void doHead(HttpServletRequest req, HttpServletResponse res)**

It handles the HEAD request. It is invoked by the web container.

* **protected void doOptions(HttpServletRequest req, HttpServletResponse res)**

It handles the OPTIONS request. It is invoked by the web container.

* **protected void doPut(HttpServletRequest req, HttpServletResponse res)**

It handles the PUT request. It is invoked by the web container.

* **protected void doTrace(HttpServletRequest req, HttpServletResponse res)**

It handles the TRACE request. It is invoked by the web container.

* **protected void doDelete(HttpServletRequest req, HttpServletResponse res)**

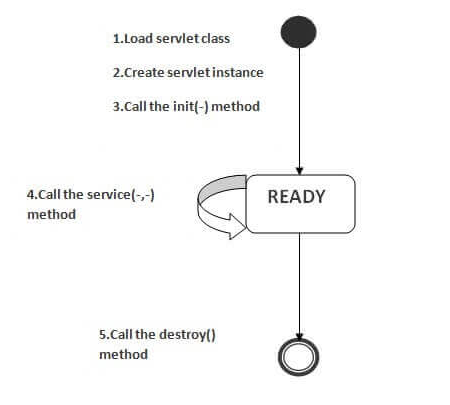
It handles the DELETE request. It is invoked by the web container.

* **protected long getLastModified(HttpServletRequest req)**

It returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

**6.. Life Cycle of Servlet**

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



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:

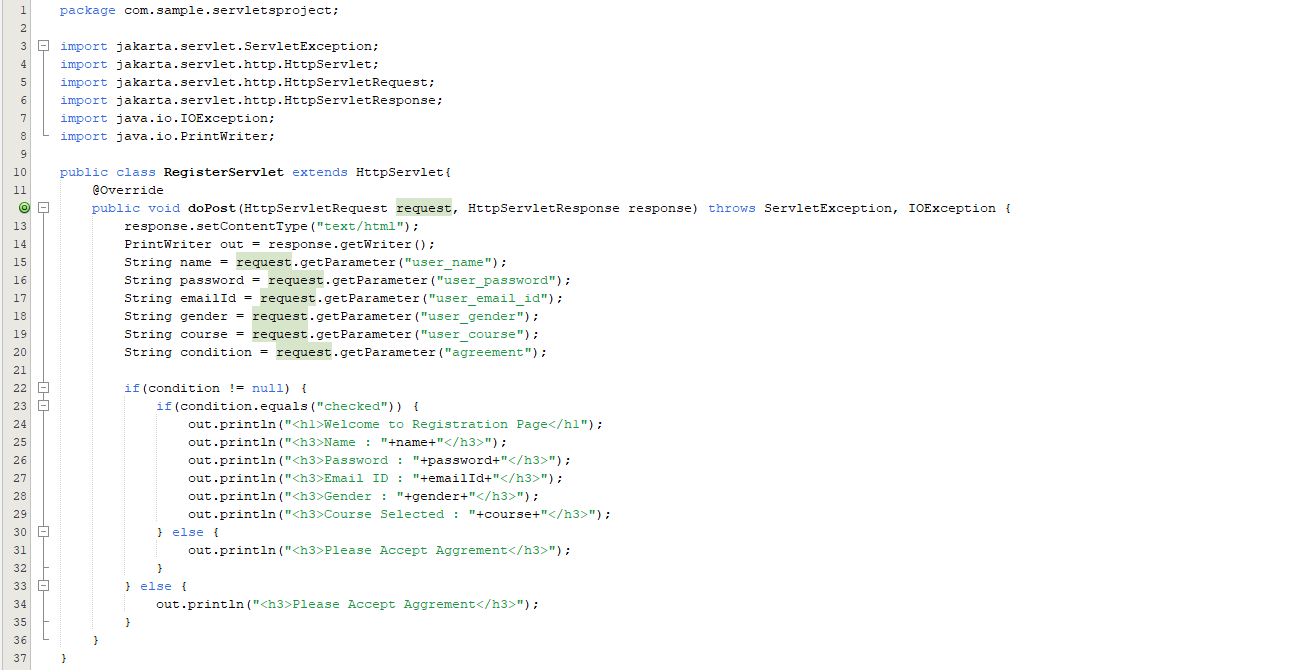
*public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException*

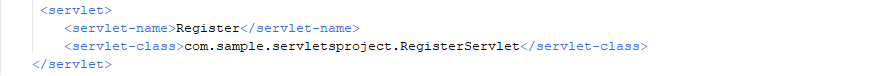
**5) destroy method is invoked**

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()

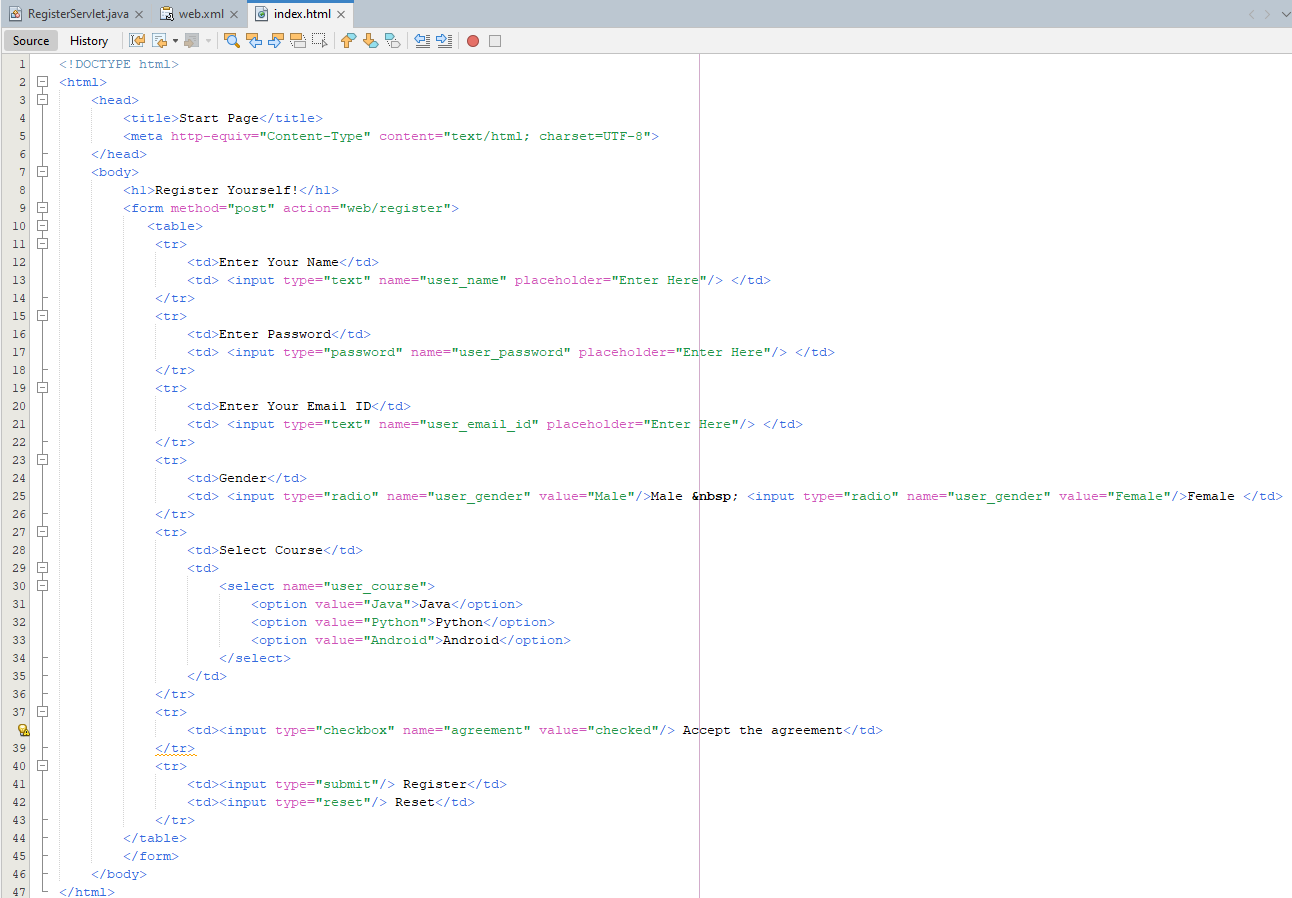
**7.. Creating Simple Registration Form using Servlet**



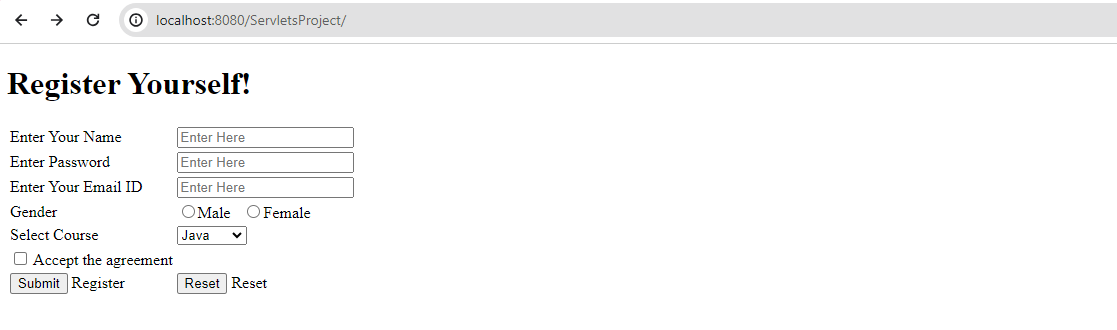




Design a Registration Form in index.html:



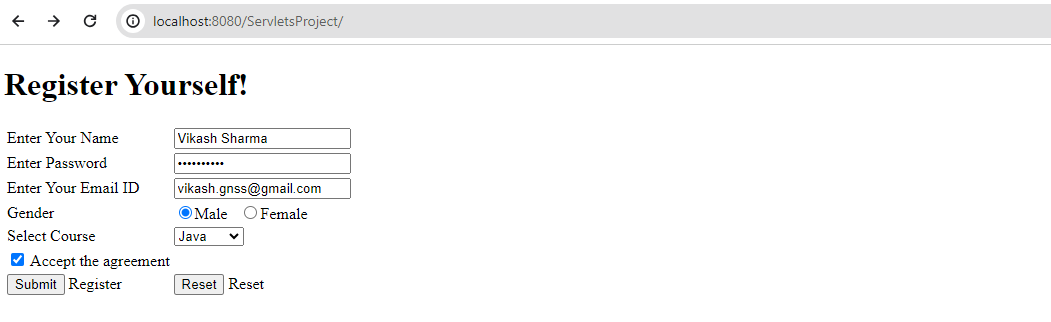
Now, let’s run the application.

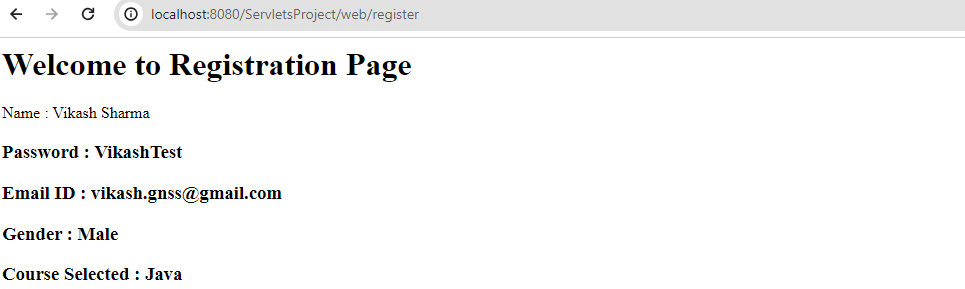


Now without checking Check box click on submit:



Now, let’s input some values and Submit again.





Here, we are just getting the values entered into the form and printing them on page. We can extend the functionality by configuring database and saving values into database.

Note: The name attribute of form control should be used as an argument of getParameter() method of HttpServletRequest class.

**8.. Internal Working of Servlet**

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

The server checks if the servlet is requested for the first time. If yes, web container does the following tasks:

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

else

* calls the service method passing request and response objects

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

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

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

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



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



**9.. Deployment Descriptor File (web.xml)**

This is a file which contains the configuration of Java Web Application. It resides in the WEB-INF folder of our Java Web Application. This is how the web.xml file looks like:



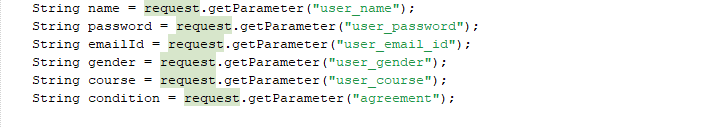
Between <web-app></web-app> tags, we can declare Servlets, we can map the Servlets, we can create Listeners, filters, initialization parameters etc.

**10.. Parameters and Attributes in Servlets**

**10.1 Parameters**

* Parameters are those values which are provided by user to any servlet to process the request during the request operation.
* Servlet only ready that value for request processing.
* Parameter is mostly sent by user via HTML forms or via web.xml file. Parameter declared in web.xml file is called initialization parameter.
* To get initialization parameter we have to pass that parameter name as an argument of getParameter() method of HttpServletRequest class.

Example:



**10.2 Attributes**

An attribute in servlet is an object that can be set, get or removed from one of the following scopes:

* request scope
* session scope
* application scope

The servlet programmer can pass information from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again. Hence, attributes are the objects that are attached by one Servlet like Session, request, config, context etc and other servlet can fetch that objects to process the logic.

Servlets can easily add, remove and modify the content of attribute whenever it is required.

**Attribute specific methods of ServletRequest, HttpSession and ServletContext interface**

There are following 4 attribute specific methods. They are as follows:

* **public void setAttribute(String name, Object object)**

sets the given object in the application scope.

* **public Object getAttribute(String name)**

Returns the attribute for the specified name.

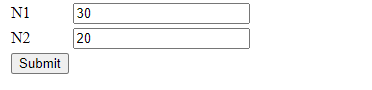
* **public Enumeration getInitParameterNames()**

Returns the names of the context's initialization parameters as an Enumeration of String objects.

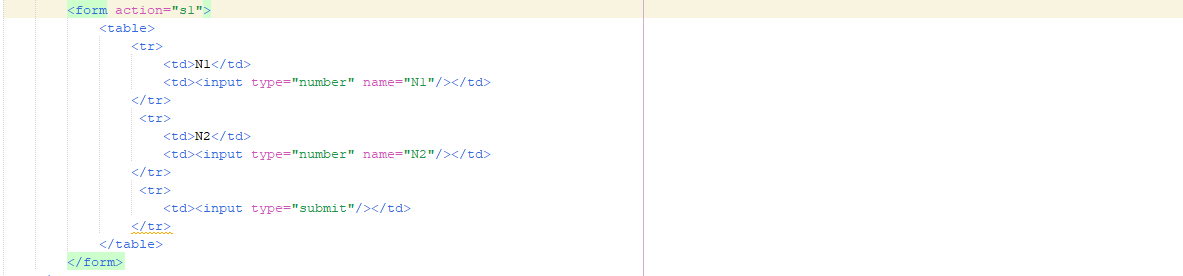
* **public void removeAttribute(String name)**

Removes the attribute with the given name from the servlet context.

**Example:**



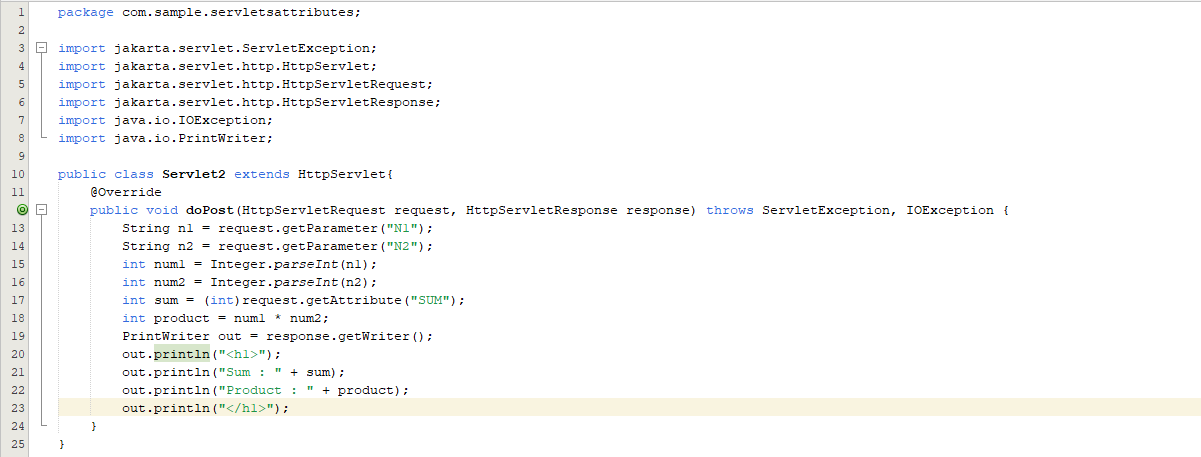
**Index.html File**



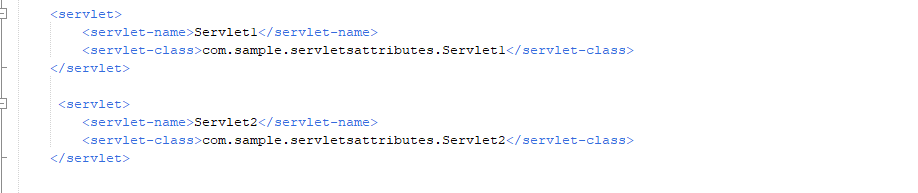
**Servlet1**

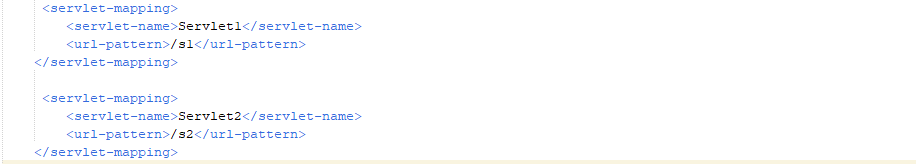


**Servlet2**



**Web.xml File**





**Output**



**11..**