



Software development Lab

Web services

Name: Manikandan P
RegNo: 2019202030

Introduction:

Web services include any software, application, or cloud technology that provides standardized web protocols (HTTP or HTTPS) to interoperate, communicate, and exchange data messaging – usually XML (Extensible Markup Language) – throughout the internet.

A web service supports communication among numerous apps with HTML, XML, WSDL, SOAP, and other open standards. XML tags the data, SOAP transfers the message, and WSDL describes the service's accessibility.

Pre-requisites:

- NetBeans IDE 6.0 Web & Java EE.
- Java Standard Development Kit (JDK) version 5.0 or version 6.0.
- Glassfish V2 or Tomcat Web Server, both of which you can select from the installer that you use to install NetBeans IDE 6.0.

Creating a Web Service

The goal of this exercise is to create a project appropriate to the deployment container that you decide to use. Once you have a project, you will create a web service in it.

Choosing a Container

You can either deploy your web service in a web container or in an EJB container. This depends on implementation choices. For example, if you plan to deploy to the Tomcat Web Server, which only has a web container, you should choose to create a web application, and not an EJB module.

- Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category or EJB Module from the Enterprise category.
- Name the project CalculatorWSApplication.
- Depending on the deployment server that you want to use, do the following: For GlassFish, set the Java EE Version to Java EE 5.
- For the Tomcat Web Server, unselect the Set Source Level to 1.4 checkbox.
- Click Finish.

Creating a Web Service from a Java Class

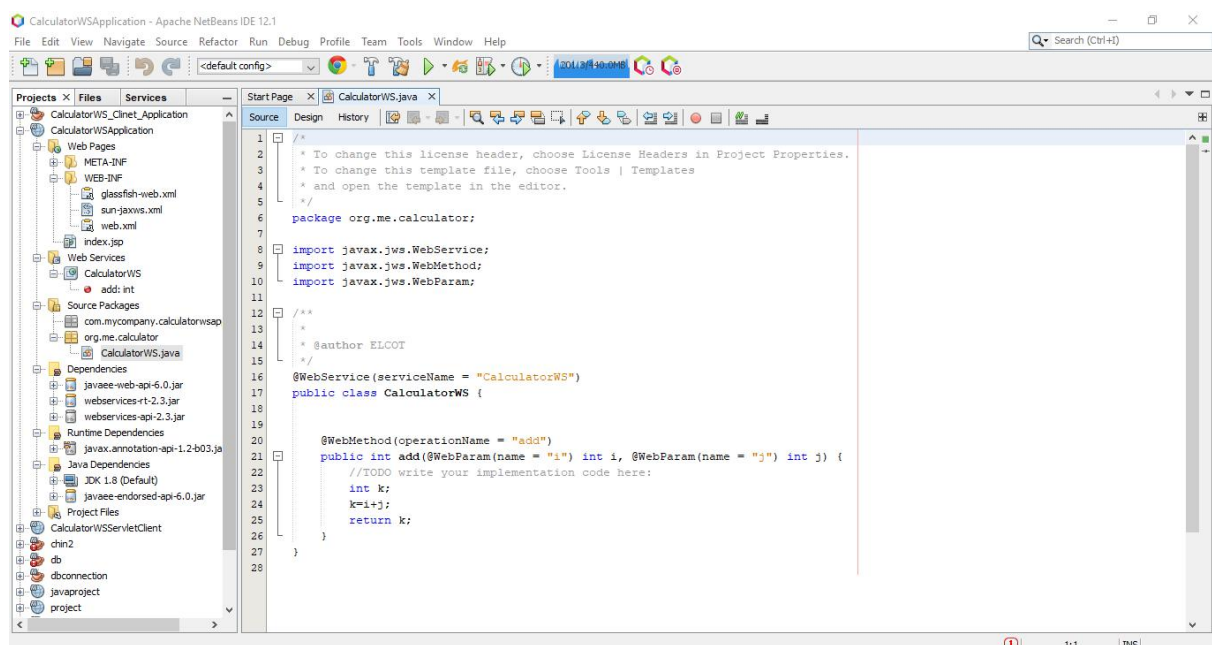
- Right-click the CalculatorWSApplication node and choose New > Web Service.
- Name the web service CalculatorWS, type org.me.calculator in Package, and click Finish.

Designing the Web Service

The goal of this exercise is to do something meaningful with the files and code that the IDE has generated for you. You will add an operation that will add two numbers received from a client.

Adding Business Logic to the Web Service

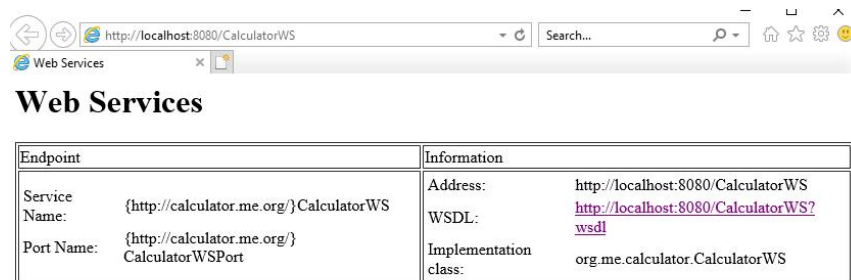
1. Click Add Operation in the visual designer. A dialog box appears, where you can define the new operation.
2. In the upper part of the Add Operation dialog box, type add in Name and type int in the Return Type drop-down list. In the lower part of the Add Operation dialog box, click Add and create a parameter of type int named i. Then click Add again and create a parameter of type int called j.
3. Click OK at the bottom of the Add Operation dialog box.
4. Click Source and notice that the source code.
5. In the editor, extend the skeleton add operation to the following:



Deploying and Testing the Web Service:

When you deploy a web service to a web container, the IDE lets you test the web service to see if it functions as you expect. The Tester application, provided by the Sun Java System Application Server, is integrated into the IDE for this purpose. For the Tomcat Web Server, there is a similar tool. However, while the Sun Java System Application Server's Tester page lets you enter values and test them, the Tomcat Web Server does not. In the latter case, you can only see that the web service is deployed, you cannot test the values. No facility for testing whether an EJB module is deployed successfully is currently available.

To test successful deployment to a web container:



Endpoint		Information	
Service Name:	{http://calculator.me.org/} CalculatorWS	Address:	http://localhost:8080/CalculatorWS
		WSDL:	http://localhost:8080/CalculatorWS?wsdl
Port Name:	{http://calculator.me.org/} CalculatorWSPort	Implementation class:	org.me.calculator.CalculatorWS

Right-click the project node, choose Properties, and click Run. Depending on the deployment server that you want to use, do the following: For the Sun Java System Application Server, type /CalculatorWSService?Tester in the Relative URL field. For the Tomcat Web Server, type /CalculatorWS?Tester in the Relative URL field.

Note: Since the result of a deployed EJB module is not displayed in a browser, you cannot take the step above if you are working with an EJB module.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000) JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision#unknown. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is Metro/2.3 (tags/2.3-7528; 2013-04-29T19:34:10+0000) JAXWS-RI/2.2.8 JAXWS/2.2 svn-revision#unknown. -->
<definitions name="CalculatorWS" targetNamespace="http://calculator.me.org/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://calculator.me.org/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
  <types>
    <xsd:schema>
      <xsd:import schemaLocation="http://localhost:8080/CalculatorWS?xsd=1" namespace="http://calculator.me.org/" />
    </xsd:schema>
  </types>
  <message name="add">
    <part name="parameters" element="tns:add"/>
  </message>
  <message name="addResponse">
    <part name="parameters" element="tns:addResponse"/>
  </message>
  <portType name="CalculatorWS">
    <operation name="add">
      <input message="tns:add" wsam:Action="http://calculator.me.org/CalculatorWS/addRequest"/>
      <output message="tns:addResponse" wsam:Action="http://calculator.me.org/CalculatorWS/addResponse"/>
    </operation>
  </portType>
  <binding name="CalculatorWSPortBinding" type="tns:CalculatorWS">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="add">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="CalculatorWS">
    <port name="CalculatorWSPort" binding="tns:CalculatorWSPortBinding"/>
  </service>
</definitions>
```

Consuming the Web Service

Client 1: Java Class in Java SE Application

In this section, we create a standard Java application. The wizard that we use to create the application will also create a Java class. We will then use the IDE's tools to consume the web service that we created at the start of this tutorial.

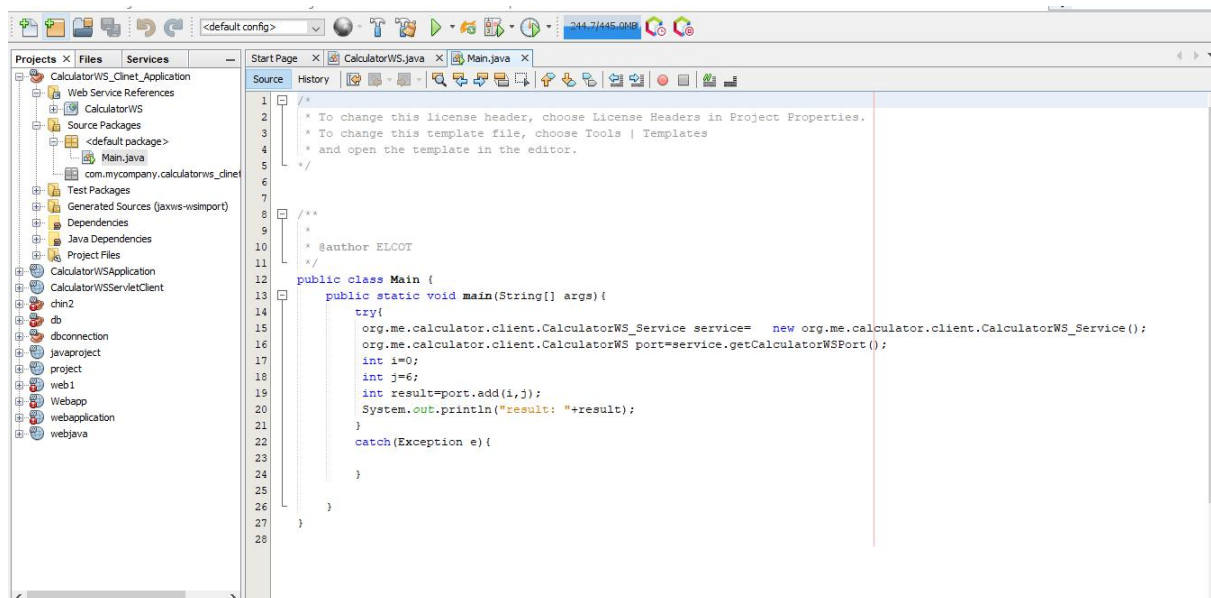
1. Choose File > New Project (Ctrl-Shift-N). Select Java Application from the General category. Name the project `CalculatorWS_Client_Application`. Click Finish.

2. Right-click the `CalculatorWS_Client_Application` node and choose New > Web Service Client.

3. In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK.

4. Type `org.me.calculator.client` in Package, and click Finish. The Projects window displays the new web service client, with a node for the add method that you created:

5. Double-click `Main.java` so that it opens in the Source Editor. Delete the TODO comment and then drag the add node above into the empty line. You should now see the following:

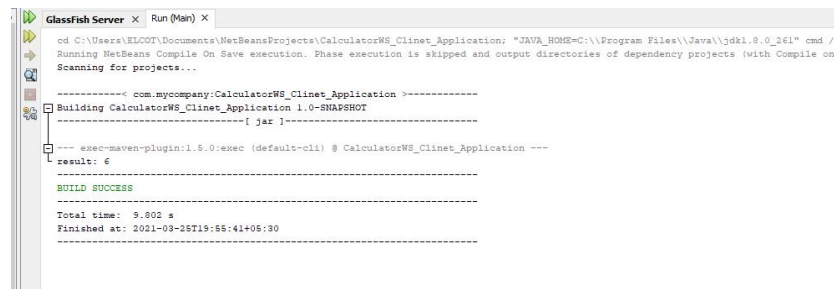


Note: Alternatively, instead of dragging the add node, you can right-click in the editor and then choose Web Service ClientResources > Call Web Service Operation.

5. Initialize the two ints, using meaningful numbers, such as 3 and 4. Just change the values of the two ints above from 0 to some other number.

7. Right-click the project node and choose Run Project.

The Output window should now show the sum:



```
cd C:\Users\ELCOT\Documents\NetBeansProjects\CalculatorWS_Client_Application; "JAVA_HOME=C:\Program Files\Java\jdk1.8.0_261" cmd /c
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of dependency projects (with Compile on
Scanning for projects...

-----< com.mycospany:CalculatorWS_Client_Application >-----
Building CalculatorWS_Client_Application 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:1.5.0:exec (default-cli) @ CalculatorWS_Client_Application ---
result: 6
BUILD SUCCESS
Total time: 9.802 s
Finished at: 2021-03-25T19:55:41+05:30
```

Client 2: Servlet in Web Application

In this section, we create a new web application, after which we create a servlet. We then use the servlet to consume the web service.

1. Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category. Name the project CalculatorWSServletClient. Click Finish.

2. Right-click the CalculatorWSServletClient node and choose New > Web Service Client. The New Web Service Client wizard appears.

2. In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK.

3. In Package, type org.me.calculator.client and click finish.

4. Right-click the CalculatorWSServletClient project node and choose New > Servlet. Name the servlet ClientServlet and place it in a package called org.me.calculator.client. Click Finish.

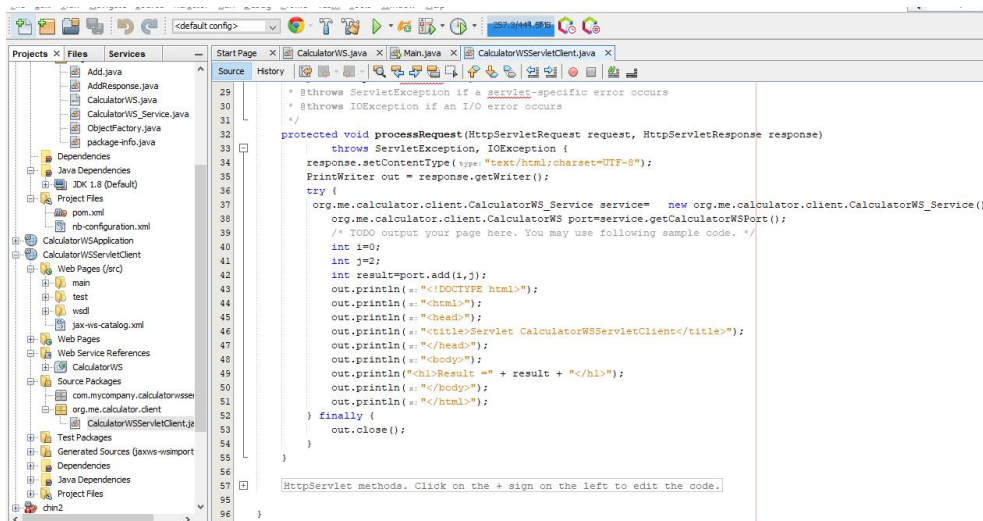
5. To make the servlet the entry point to your application, right-click the project node, choose Properties, click Run, and type /ClientServlet in Relative URL. Click OK.

6. In the Source Editor, remove the line that comments out the body of the processRequest method. This is the line:

```
/* TODO output your page here
```

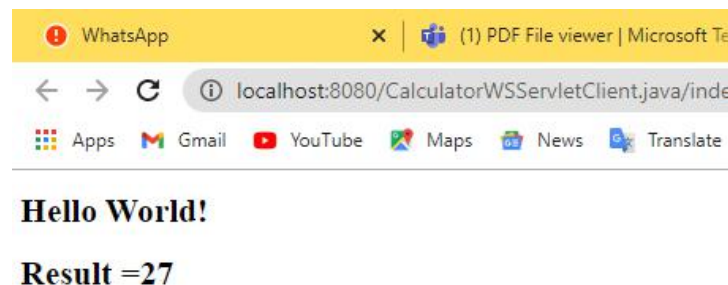
Next, delete the line that ends the section of commented out code:

```
*/
```



Add some empty lines after this line: `out.println("<h1>Servlet ClientServlet at " + request.getContextPath() + "/h1>");` Now, drag the node that represents the add operation into the space that you created. The processRequest method now looks as follows:

8. Right-click the project node and choose Run Project. The server starts, if it wasn't running already; the application is built and deployed, and the browser opens, displaying the calculation result, as shown below:



Client 3: JSP Page in Web Application:

In this section, we create a new web application and then consume our web service in the default JSP page that the Web Application wizard creates.

1. Choose File > New Project (Ctrl-Shift-N). Select Web Application from the Web category. Name the project CalculatorWSJSPClient. Click Finish.

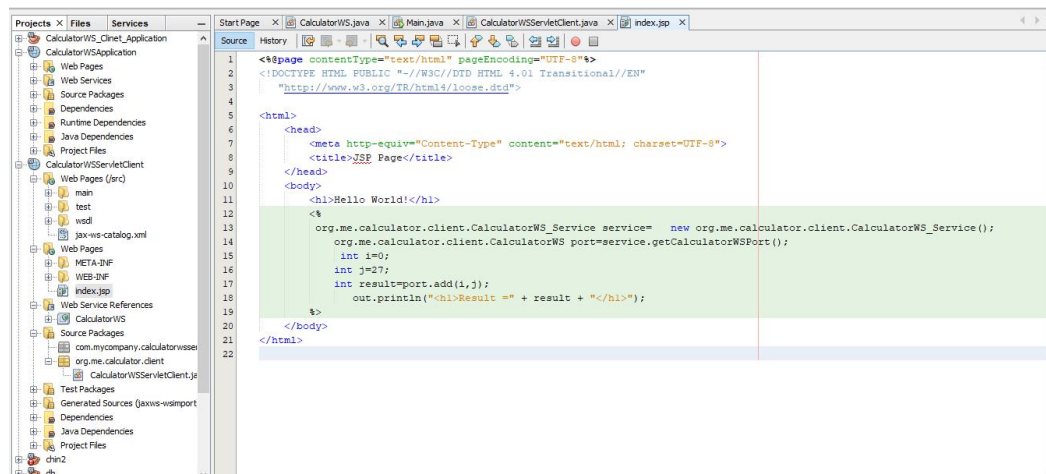
2. Right-click the CalculatorWSJSPClient node and choose New > Web Service Client.

3. In Project, click Browse. Browse to the web service that you want to consume. When you have selected the web service, click OK.

4. In Package, type org.me.calculator.client and click Finish.

5. In the Web Service References node, expand the node that represents the web service. The add operation, which you want to invoke from the client, is now exposed.

6. Drag the add operation to the client's index.jsp page, and drop it below the H1 tags. The code for invoking the service's operation is now generated in the index.jsp page, as you can see here:



```
1 <!--page contentType="text/html" pageEncoding="UTF-8"%-->
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3 "http://www.w3.org/TR/html4/loose.dtd">
4
5 <html>
6 <head>
7 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8 <title>JSP Page</title>
9 </head>
10 <body>
11 <h1>Hello World!</h1>
12
13 <%
14 org.me.calculator.client.CalculatorWS_Service service= new org.me.calculator.client.CalculatorWS_Service();
15 org.me.calculator.client.CalculatorWS port=service.getCalculatorWSPort();
16 int i=0;
17 int j=27;
18 int result=port.add(i,j);
19 out.println("<h1>Result =" + result + "</h1>");
20
21 </body>
22 </html>
```

7. Right-click the project node and choose Run Project.

The server starts, if it wasn't running already; the application is built and deployed, and the browser opens, displaying the calculation result:

