

## Getting Started with JAX-WS Web Services

This lab demonstrates the basics of using the IDE to develop a JAX-WS web service. After you create the web service, you write three different web service clients that use the web service over a network, which is called "consuming" a web service. The three clients are a Java class in a Java SE application, a servlet, and a JSP page in a web application.

### 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.

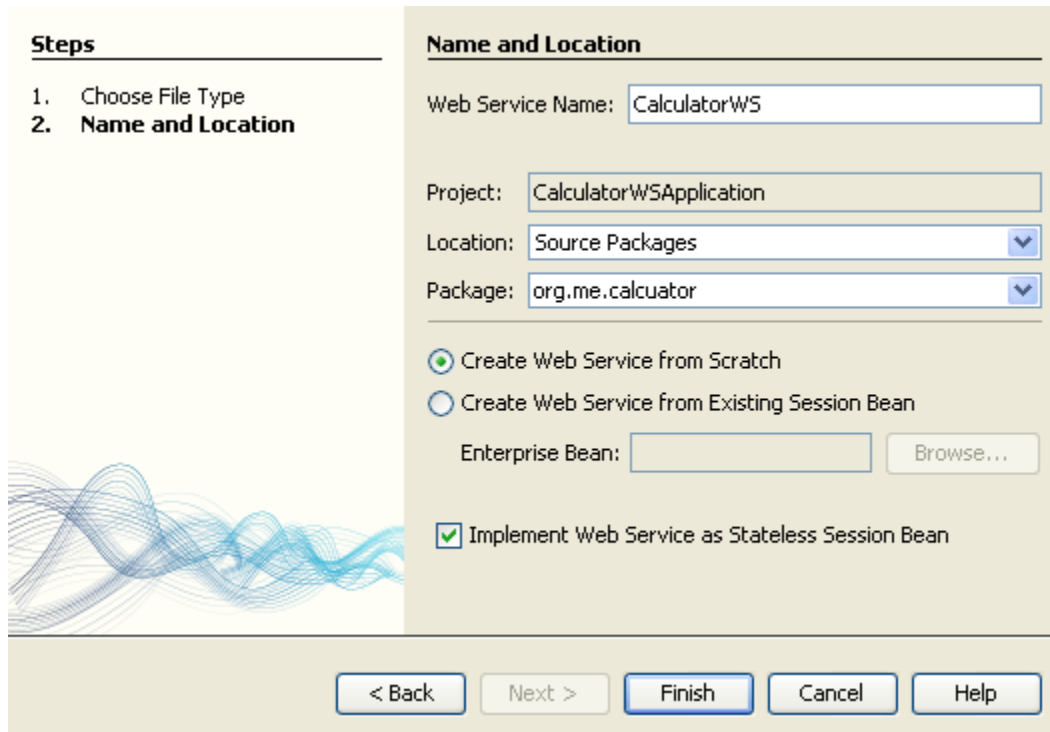
#### Choosing a Container

Create a web application.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows). Select Web Application from the Java Web category or EJB Module from the Java EE category.
2. Name the project `CalculatorWSApplication`. Select a location for the project. Click Next.
3. Select your server and Java EE version and click Finish.

#### Creating a Web Service from a Java Class

1. Right-click the `CalculatorWSApplication` node and choose New > Web Service.
2. Name the web service `CalculatorWS` and type `org.me.calculator` in Package. Leave Create Web Service from Scratch selected.
3. If you are creating a Java EE 6 project on GlassFish or WebLogic, select Implement Web Service as a Stateless Session Bean.



**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Web Service Name:

Project:

Location:  ▼

Package:  ▼

☒ Create Web Service from Scratch

☐ Create Web Service from Existing Session Bean

Enterprise Bean:

☒ Implement Web Service as Stateless Session Bean

< Back   Next >   **Finish**   Cancel   Help

4. Click Finish. The Projects window displays the structure of the new web service and the source code is shown in the editor area.

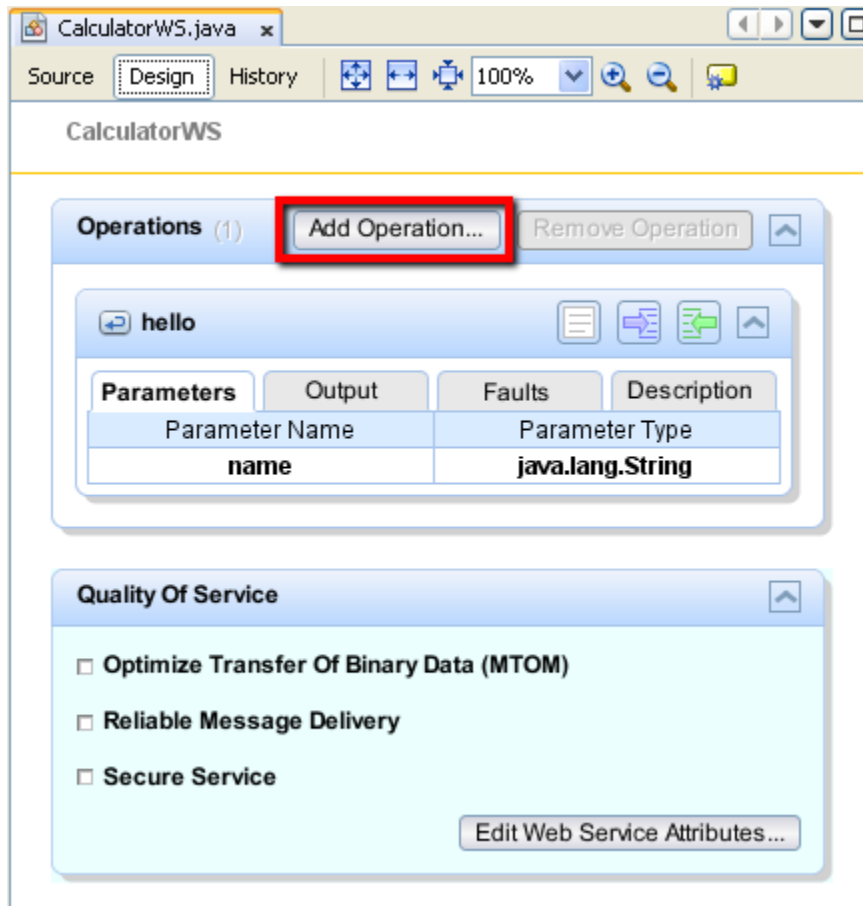
## Adding an Operation to the Web Service

The goal of this exercise is to add to the web service an operation that adds two numbers received from a client. The NetBeans IDE provides a dialog for adding an operation to a web service.

**To add an operation to the web service:**

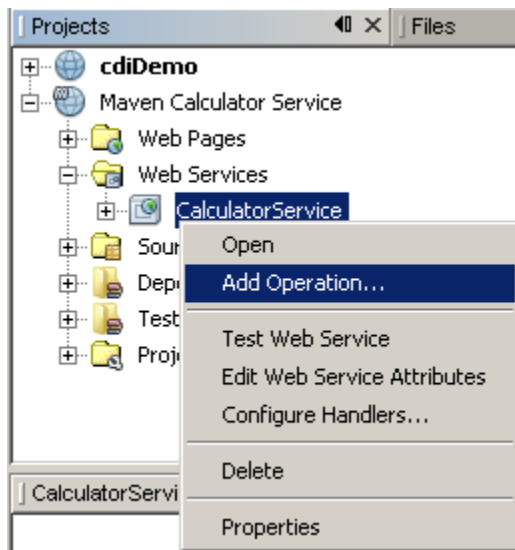
1. Either:

- Change to the Design view in the editor.



Or:

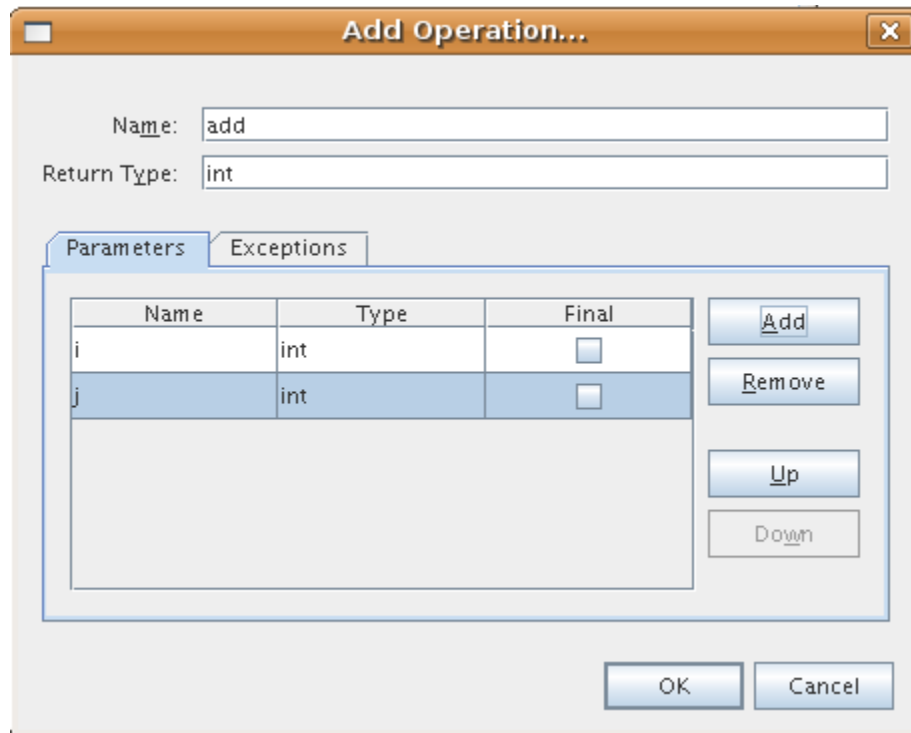
- Find the web service's node in the Projects window. Right-click that node. A context menu opens.



2. Click Add Operation in either the visual designer or the context menu. The Add Operation dialog opens.

3. In the upper part of the Add Operation dialog box, type `add` in Name and type `int` in the Return Type drop-down list.
4. In the lower part of the Add Operation dialog box, click Add and create a parameter of type `int` named `i`.
5. Click Add again and create a parameter of type `int` called `j`.

You now see the following:



6. Click OK at the bottom of the Add Operation dialog box. You return to the editor.
7. Remove the default `hello` operation, either by deleting the `hello()` method in the source code or by selecting the `hello` operation in the visual designer and clicking Remove Operation.

The visual designer now displays the following:

index.jsp x

CalculatorWS.java x

Source

Design

100%

CalculatorWSService

Operations (1)

Add Operation...

Remove Operation

add

Parameters

Output

Faults

Description

Parameter Name	Parameter Type
i	int
j	int

Quality Of Service

☐ Optimize Transfer Of Binary Data (MTOM)

☐ Reliable Message Delivery

☐ Secure Service

Advanced ...

8. Click Source and view the code that you generated in the previous steps.

```
package org.me.calculator;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

/**
 *
 * @author gw152771
 */
@WebService()
public class CalculatorWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i")
        int i, @WebParam(name = "j")
        int j) {
        //TODO write your implementation code here:
        return 0;
    }
}
```

```

package org.me.calculator;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;

/**
 *
 * @author jeff
 */
@WebService()
@Stateless()
public class CalculatorWS {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "add")
    public int add(@WebParam(name = "i")
    int i, @WebParam(name = "j")
    int j) {
        //TODO write your implementation code
        return 0;
    }
}

```

9. In the editor, extend the skeleton add operation to the following (changes are in bold):

```

@WebMethod
public int add(@WebParam(name = "i") int i, @WebParam(name = "j")
int j) {
    int k = i + j;
    return k;
}

```

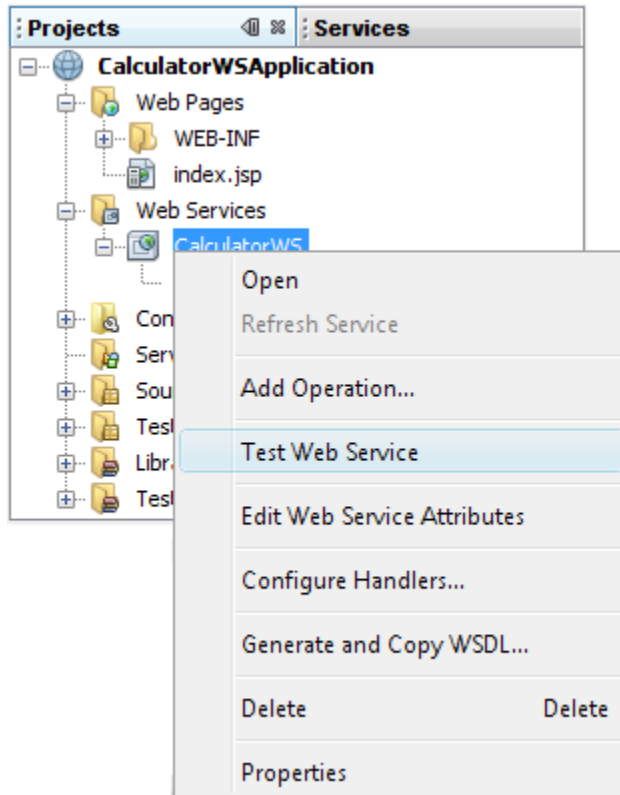
As you can see from the preceding code, the web service simply receives two numbers and then returns their sum. In the next section, you use the IDE to test the web service.

## Deploying and Testing the Web Service

After you deploy a web service to a server, you can use the IDE to open the server's test client, if the server has a test client. The GlassFish and WebLogic servers provide test clients.

**To test successful deployment to a GlassFish or WebLogic server:**

1. Right-click the project and choose Deploy. The IDE starts the application server, builds the application, and deploys the application to the server. You can follow the progress of these operations in the CalculatorWSApplication (run-deploy) and the GlassFish server or Tomcat tabs in the Output view.
2. In the IDE's Projects tab, expand the Web Services node of the CalculatorWSApplication project. Right-click the CalculatorWS node, and choose Test Web Service.



The IDE opens the tester page in your browser, if you deployed a web application to the GlassFish server.



- If you deployed to the GlassFish server, type two numbers in the tester page, as shown below:

## CalculatorWS Web Service Tester

This form will allow you to test your web service implementation ([WSDL File](#))

---

To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

### Methods :

public abstract int org.netbeans.CalculatorWSProject.add(int,int)

(  ,  )

---

The sum of the two numbers is displayed:

## add Method invocation

---

### Method parameter(s)

Type	Value
int	2
int	3

---

### Method returned

int : "5"

---



## Consuming the Web Service

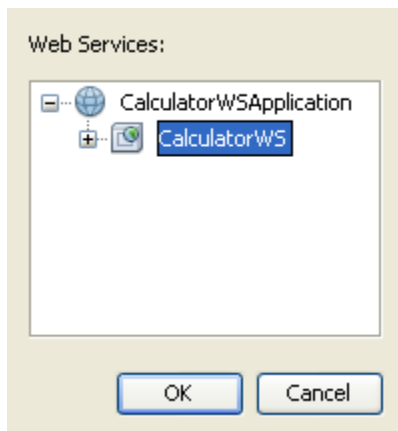
---

Now that you have deployed the web service, you need to create a client to make use of the web service's add method. Here, you create three clients— a Java class in a Java SE application, a servlet, and a JSP page in a web application.

## Client 1: Java Class in Java SE Application

In this section, you create a standard Java application. The wizard that you use to create the application also creates a Java class. You then use the IDE's tools to create a client and consume the web service that you created at the start of this tutorial.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows). Select Java Application from the Java category. Name the project `CalculatorWS_Client_Application`. Leave `Create Main Class` selected and accept all other default settings. Click Finish.
2. Right-click the `CalculatorWS_Client_Application` node and choose New > Web Service Client. The New Web Service Client wizard opens.
3. Select Project as the WSDL source. Click Browse. Browse to the `CalculatorWS` web service in the `CalculatorWSApplication` project. When you have selected the web service, click OK.



- Do not select a package name. Leave this field empty.

**Steps**

- Choose File Type
- WSDL and Client Location**

**WSDL and Client Location**

Specify the WSDL file of the Web Service.

☒ Project:

☐ Local File:

☐ WSDL URL:

Specify a package name where the client java artifacts will be gener...

Project:

Package:

Client Style:

☐ Generate Dispatch code

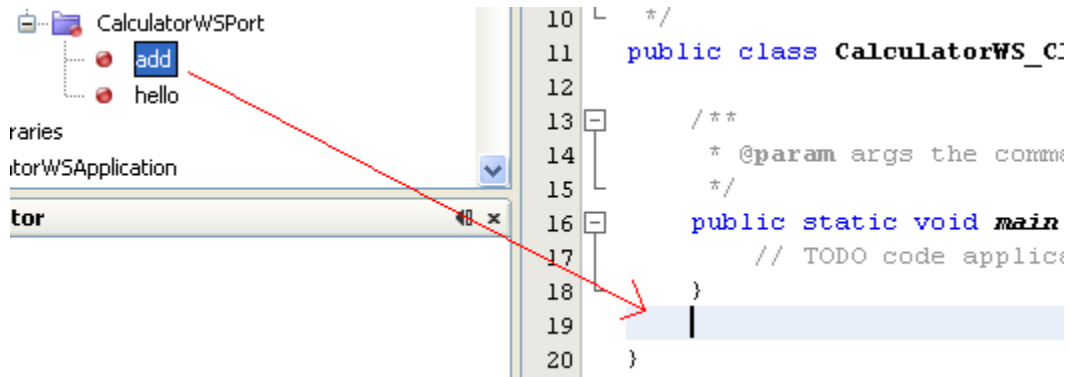
- Leave the other settings at default and click Finish.

The Projects window displays the new web service client, with a node for the add method that you created:

The screenshot shows the Eclipse IDE's Projects view. The project 'CalculatorWS\_Client\_Application' is expanded, revealing its internal structure:

- Source Packages:**
  - META-INF
  - META-INF.wsdl.localhost\_8080.CalculatorWS
  - calculatorws\_client\_application
  - CalculatorWS\_Client\_Application.java
- Generated Sources (jax-ws):**
- Web Service References:**
  - CalculatorWS
    - CalculatorWS
    - CalculatorWSPort
      - add
- Libraries:**
  - CalculatorWSApplication

- Double-click your main class so that it opens in the Source Editor. Drag the `add` node below the `main()` method.



You now see the following:

```
public static void main(String[] args) {
    // TODO code application logic here
}

private static int add(int i, int j) {
    org.me.calculator.CalculatorWS_Service service = new
org.me.calculator.CalculatorWS_Service();
    org.me.calculator.CalculatorWS port =
service.getCalculatorWSPort();
    return port.add(i, j);
}
```

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

- In the `main()` method body, replace the `TODO` comment with code that initializes values for `i` and `j`, calls `add()`, and prints the result.

```
8. public static void main(String[] args) {
    int i = 3;
    int j = 4;
    int result = add(i, j);
    System.out.println("Result = " + result);
}
```

- Surround the `main()` method code with a `try/catch` block that prints an exception.

```
public static void main(String[] args) {  
    try {  
        int i = 3;  
        int j = 4;  
        int result = add(i, j);  
        System.out.println("Result = " + result);  
    } catch (Exception ex) {  
        System.out.println("Exception: " + ex);  
    }  
}
```

10. Right-click the project node and choose Run.

The Output window now shows the sum:

```
compile:  
run:  
Result = 7  
BUILD SUCCESSFUL (total time: 1 second)
```

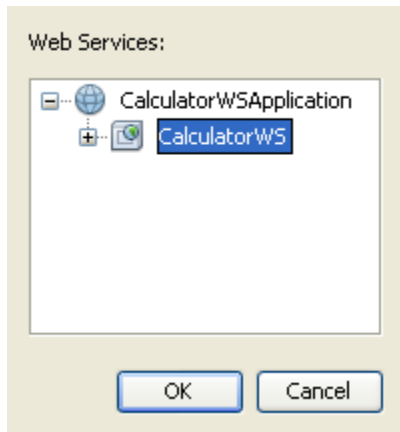
## Client 2: Servlet in Web Application

In this section, you create a new web application, after which you create a servlet. You then use the servlet to consume the web service that you created at the start of this tutorial.

1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows). Select Web Application from the Java Web category. Name the project `CalculatorWSServletClient`. Click Next and then click Finish.
2. Right-click the `CalculatorWSServletClient` node and choose New > Web Service Client.

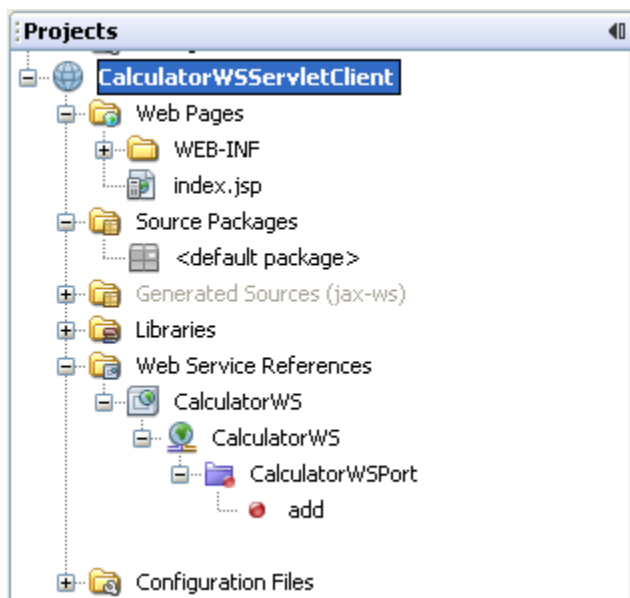
The New Web Service Client wizard appears.

3. Select Project as the WSDL source. Click Browse. Browse to the CalculatorWS web service in the CalculatorWSApplication project. When you have selected the web service, click OK.



4. Do not select a package name. Leave this field empty.
5. Leave the other settings at default and click Finish.

The Web Service References node in the Projects window displays the structure of your newly created client, including the `add` operation that you created earlier in this tutorial:



6. 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.

7. To make the servlet the entry point to your application, right-click the CalculatorWSServletClient project node and choose Properties. Open the Run properties and type /ClientServlet in the Relative URL field. Click OK.
8. If there are error icons for ClientServlet.java, right-click the project node and select Clean and Build.
9. In the Source Editor, drag the add operation anywhere in the body of the ClientServlet class. The add() method appears at the end of the class code.

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

```
private int add(int i, int j) {  
    org.me.calculator.CalculatorWS port =  
service.getCalculatorWSPort();  
    return port.add(i, j);  
}
```

10. In the processRequest() method, add some empty lines after this line:

```
out.println("<h1>Servlet ClientServlet at " +  
request.getContextPath () + "</h1>");
```

11. Add code that initializes values for i and j, calls add(), and prints the result. The added code is in **boldface**:

```
protected void processRequest(HttpServletRequest request,  
HttpServletResponse response)  
  
    throws ServletException, IOException {  
response.setContentType("text/html;charset=UTF-8");  
PrintWriter out = response.getWriter();  
try {  
    out.println("<html>");  
    out.println("<head>");  
    out.println("<title>Servlet ClientServlet</title>");  
    out.println("</head>");  
    out.println("<body>");  
    out.println("<h1>Servlet ClientServlet at " +  
request.getContextPath () + "</h1>");
```

```

        int i = 3;
        int j = 4;
        int result = add(i, j);
        out.println("Result = " + result);

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

    } finally {
        out.close();
    }
}

```

12. Surround the added code with a try/catch block that prints an exception.

```

protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet ClientServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet ClientServlet at " +
request.getContextPath () + "</h1>");

        try {
            int i = 3;
            int j = 4;
            int result = add(i, j);
            out.println("Result = " + result);
        } catch (Exception ex) {
            out.println("Exception: " + ex);
        }
    }
}

```



```
        out.println("</body>");  
        out.println("</html>");  
  
    } finally {  
        out.close();  
    }  
}
```

13. Right-click the project node and choose Run.

The server starts, the application is built and deployed, and the browser opens, displaying the calculation result, as shown below:

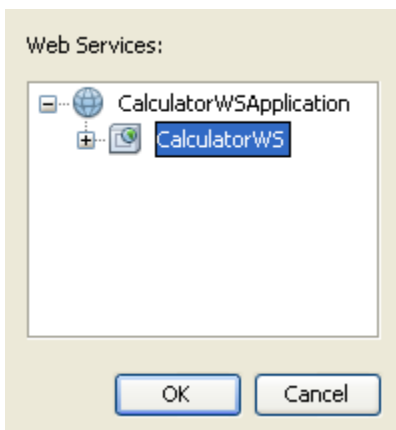
# Servlet ClientServlet at /CalculatorWSServletClient

Result = 7

## Client 3: JSP Page in Web Application

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

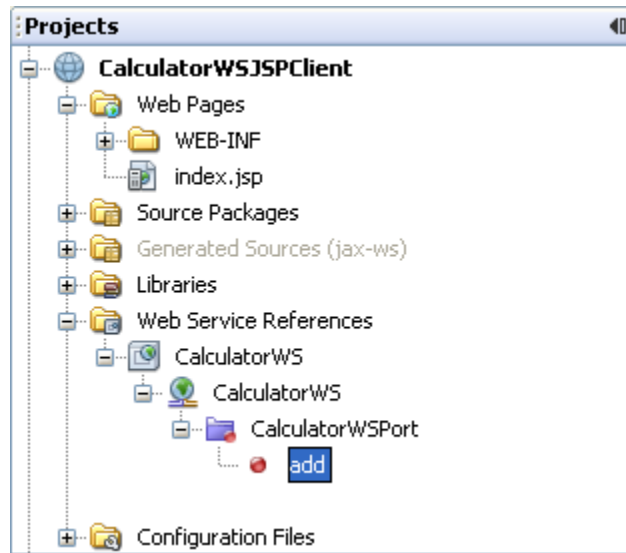
1. Choose File > New Project (Ctrl-Shift-N on Linux and Windows). Select Web Application from the Java Web category. Name the project `CalculatorWSJSPClient`. Click Finish.
2. Right-click the `CalculatorWSJSPClient` node and choose New > Web Service Client.
3. Select Project as the WSDL source. Click Browse. Browse to the `CalculatorWS` web service in the `CalculatorWSApplication` project. When you have selected the web service, click OK.



4. Do not select a package name. Leave this field empty.

5. Leave the other settings at default and click Finish.

The Projects window displays the new web service client, as shown below:



6. In the Web Service References node, expand the node that represents the web service. The add operation, which you will invoke from the client, is now exposed.
7. 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:

```
<%  
  
try {  
    org.me.calculator.CalculatorWSService service = new  
org.me.calculator.CalculatorWSService();  
    org.me.calculator.CalculatorWS port =  
service.getCalculatorWSPort();  
    // TODO initialize WS operation arguments here  
    int i = 0;  
    int j = 0;  
    // TODO process result here  
    int result = port.add(i, j);  
    out.println("Result = "+result);  
} catch (Exception ex) {  
    // TODO handle custom exceptions here  
}
```

%>

Change the value for `i` and `j` from 0 to other integers, such as 3 and 4. Replace the commented out TODO line in the catch block with `out.println("exception" + ex);`.

8. Right-click the project node and choose Run.

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

## JSP Page

---

Result = 7

---