ASSIGNMENT NO. 5	
Assignment No. 5	To create a simple **web service** and write any distributed application to consume the web service.
Objective(s):	By the end of this assignment, the student will be able to create, deploy and test Web-service application.
Tools	NetBeans IDE with GlassFish Server, Java 8

WEB-SERVICES

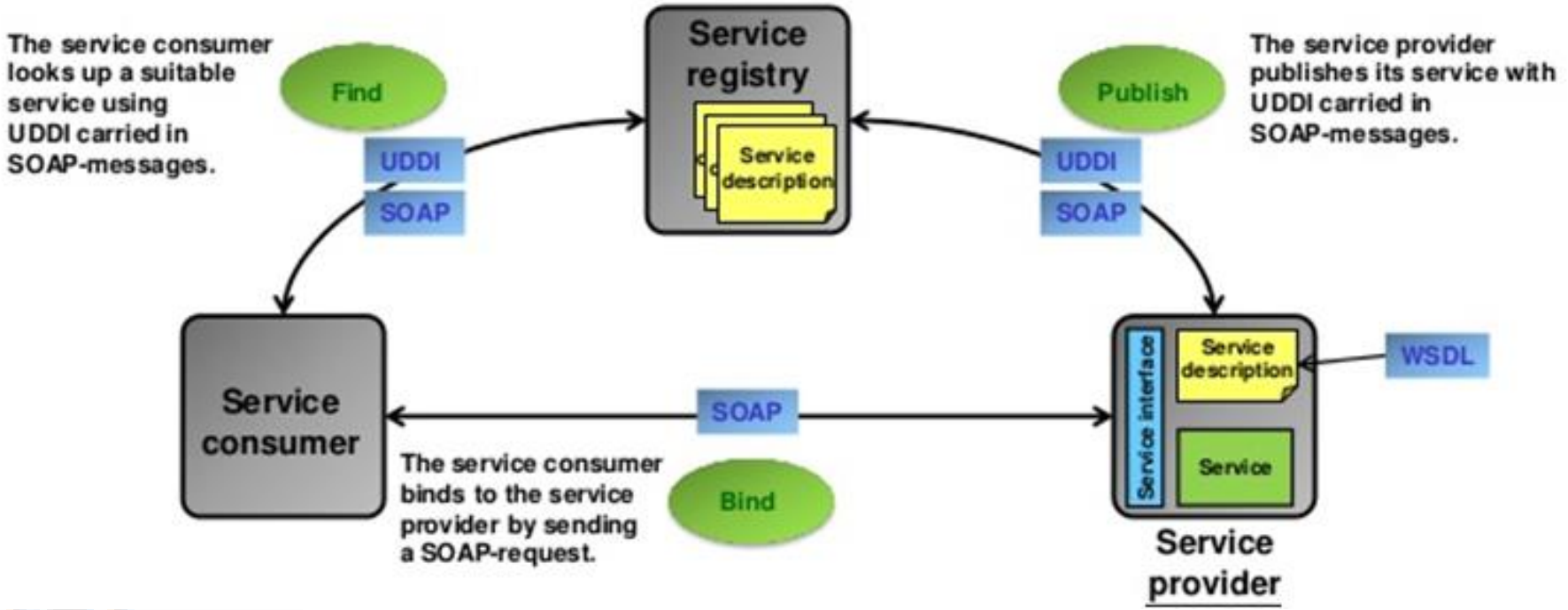
- A Web service, is a method of **communication between two applications** or electronic devices **over the World Wide Web** (WWW).
- A web service can be defined as a **collection of open protocols and standards for exchanging information among systems or applications.**

- A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided.

WEB SERVICE ARCHITECTURES

The combo SOAP+WSDL+UDDI defines a general model for a web service architecture.

SOAP:	Simple Object Access Protocol
WSDL:	Web Service Description Language
UDDI:	Universal Description and Discovery Protocol
Service consumer:	User of a service
Service provider:	Entity that implements a service (=server)
Service registry:	Central place where available services are listed and advertised for lookup



TYPES OF WEB SERVICES

There are two types of web services:

SOAP-Based Web Services:

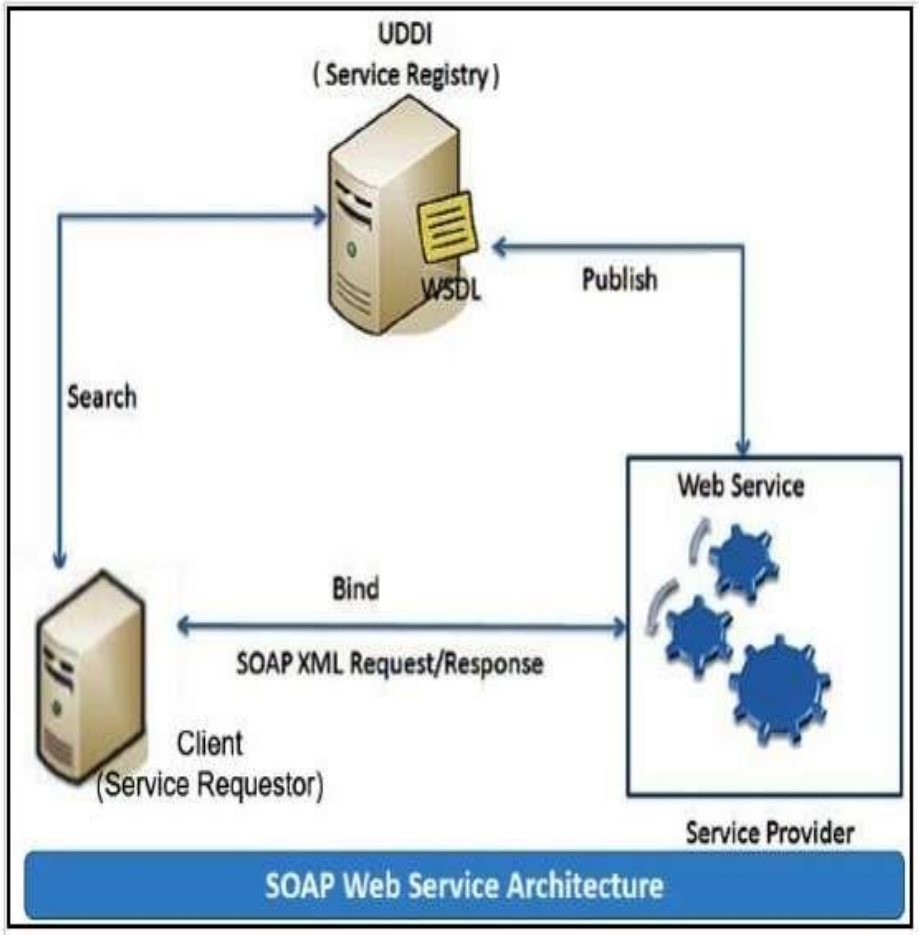
SOAP stands for Simple Object Access Protocol. SOAP is an XML based industry standard protocol for designing and developing web services. Since it's XML based, it's platform and language independent. So, our server can be based on JAVA and client can be on .NET, PHP etc. and vice versa.

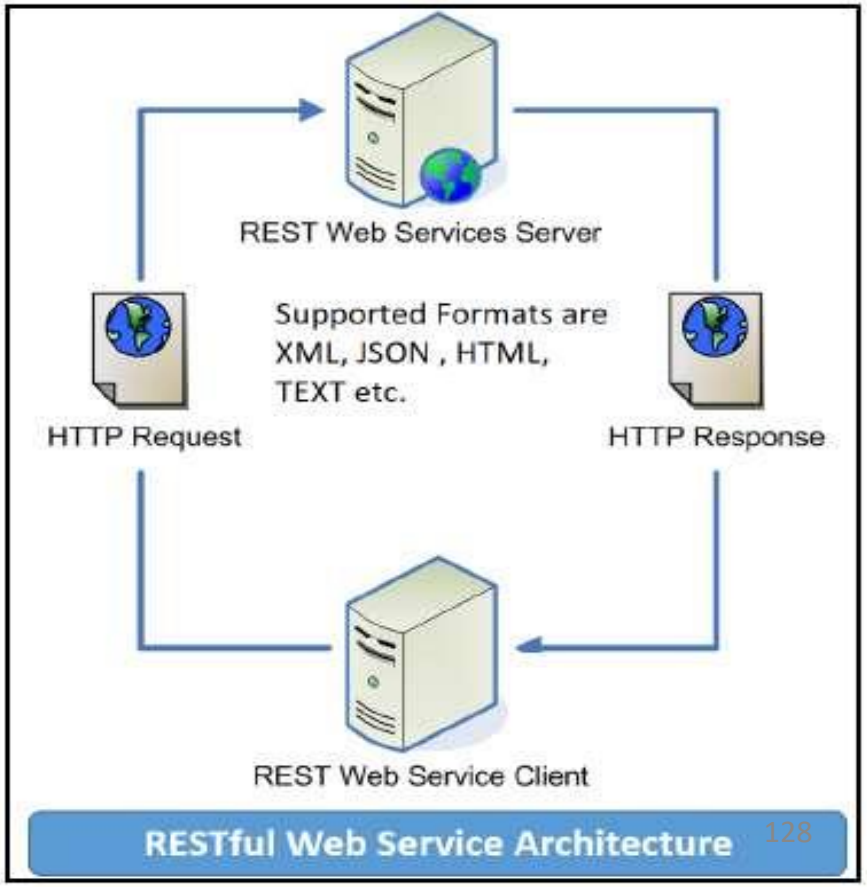
RESTful Web Services-

REST (Representational State Transfer) is an architectural style for developing web services. It's getting popularity recently because it has small learning curve when compared to SOAP. Resources are core concepts of Restful web services and they are uniquely identified by their URIs

SOAP BASED WEB SERVICES Vs. RESTful Web Services

REST stands for **Representational State Transfe**r. RESTful web services are considered a performance-efficient alternative to the SOAP web services. REST is an architectural style, not a protocol.





SOAP vs REST

SOAP	REST
SOAP is a protocol.	REST is an architectural style.
SOAP stands for Simple Object Access Protocol.	REST stands for REpresentational State Transfer.
SOAP can't use REST because it is a protocol.	REST can use SOAP web services because it is a concept and can use any protocol like HTTP, SOAP.
SOAP uses services interfaces to expose the business logic.	REST uses URI to expose business logic.
JAX-WS is the java API for SOAP web services.	JAX-RS is the java API for RESTful web services.
SOAP defines standards to be strictly followed.	REST does not define too much standards like SOAP.
SOAP requires more bandwidth and resource than REST.	REST requires less bandwidth and resource than SOAP.
SOAP defines its own security.	RESTful web services inherits security measures from the underlying transport.
SOAP permits XML data format only.	REST permits different data format such as Plain text, HTML, XML, JSON etc.
SOAP is less preferred than REST.	REST more preferred than SOAP.

STEPS INVOLVED IN BASIC SOAP WEB SERVICE OPERATIONAL BEHAVIOUR

The following are the steps involved in a basic SOAP web service operational behavior:

1. The client program that wants to interact with another application prepares its request content as a SOAP message.

2. Then, the client program sends this SOAP message to the server web service as an HTTP POST request with the content passed as the body of the request.

3. The web service plays a crucial role in this step by understanding the SOAP request and converting it into a set of instructions that the server program can understand.

4. The server program processes the request content as programmed and prepares the output as the response to the SOAP request.

5. Then, the web service takes this response content as a SOAP message and reverts to the SOAP HTTP request invoked by the client program with this response.

6. The client program web service reads the SOAP response message to receive the outcome of the server program for the request content it sent as a request.

DESIGNING THE SOLUTION

Java provides it's own API to create both SOAP as well as RESTful web services.

1. JAX-WS: JAX-WS stands for Java API for XML Web Services. JAX-WS is XML based Java API to build web services server and client application.

2. JAX-RS: Java API for RESTful Web Services (JAX-RS) is the Java API for creating REST web services. JAX-RS uses annotations to simplify the development and deployment of web services.

Both of these APIs are part of standard JDK installation, so we don't need to add any jars to work with them.

Students are required to implement both i.e. using SOAP and RESTful APIs.

WEB-SERVICES IMPLEMENTATION

- **Steps to implement Web-Services:**

1. Choosing container [Create a Project in Netbeans +Glassfish Server / Eclipse].
2. Creating Web Service from java class
3. Adding an Operation(Methods) to Web Service .
4. Deploying and Testing Web Service.
5. Consuming the Web Service (Creating a client and using web-service).

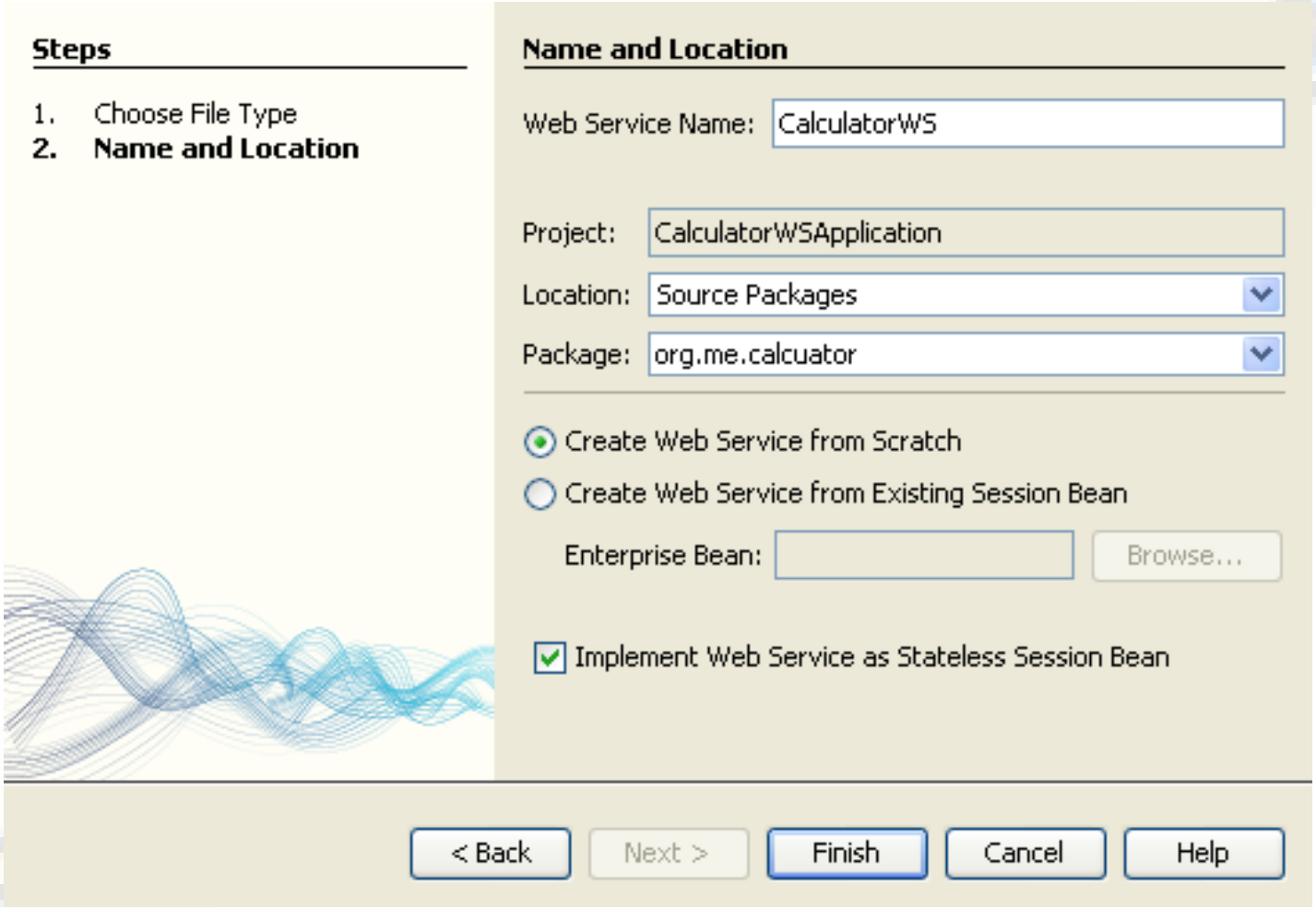
WEB-SERVICES IMPLEMENTATION

- **Step 1: Choosing a container :** You can either deploy your web service in a web container.
- Choose **File** > **New Project** (Ctrl-Shift-N on Linux and Windows).
- Select **Web Application** from the Java **Web** category.
- Name the project `CalculatorWSApplication`.
- Select a location for the project. Click **Next**.
- Select the server **[Glassfish / Tomcat]** and **Java EE** version and click **Finish.**

WEB-SERVICES IMPLEMENTATION

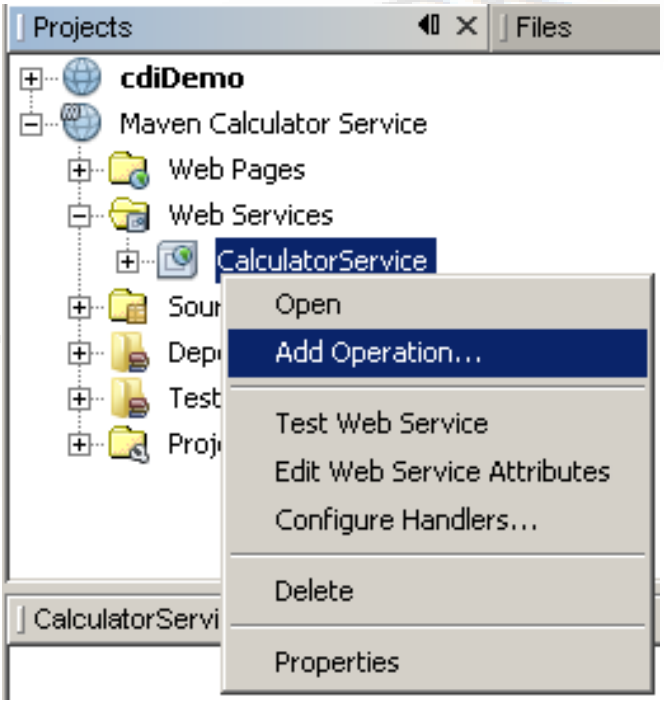
- **Step 2: Creating a Web Service from a Java Class :**
- Right-click the `CalculatorWSApplication` node and **choose New > Web Service**.
- Name the web service `CalculatorWS` and type `org.me.calculator` in Package.
- Keep "**Create Web Service from Scratch**" check box selected.
- If you are creating a Java EE project on GlassFish, select "**Implement Web Service as a Stateless Session Bean"**.
- Click **Finish**. The Projects window displays the structure of the new web service and the source code is shown in the editor area.

WEB-SERVICES IMPLEMENTATION



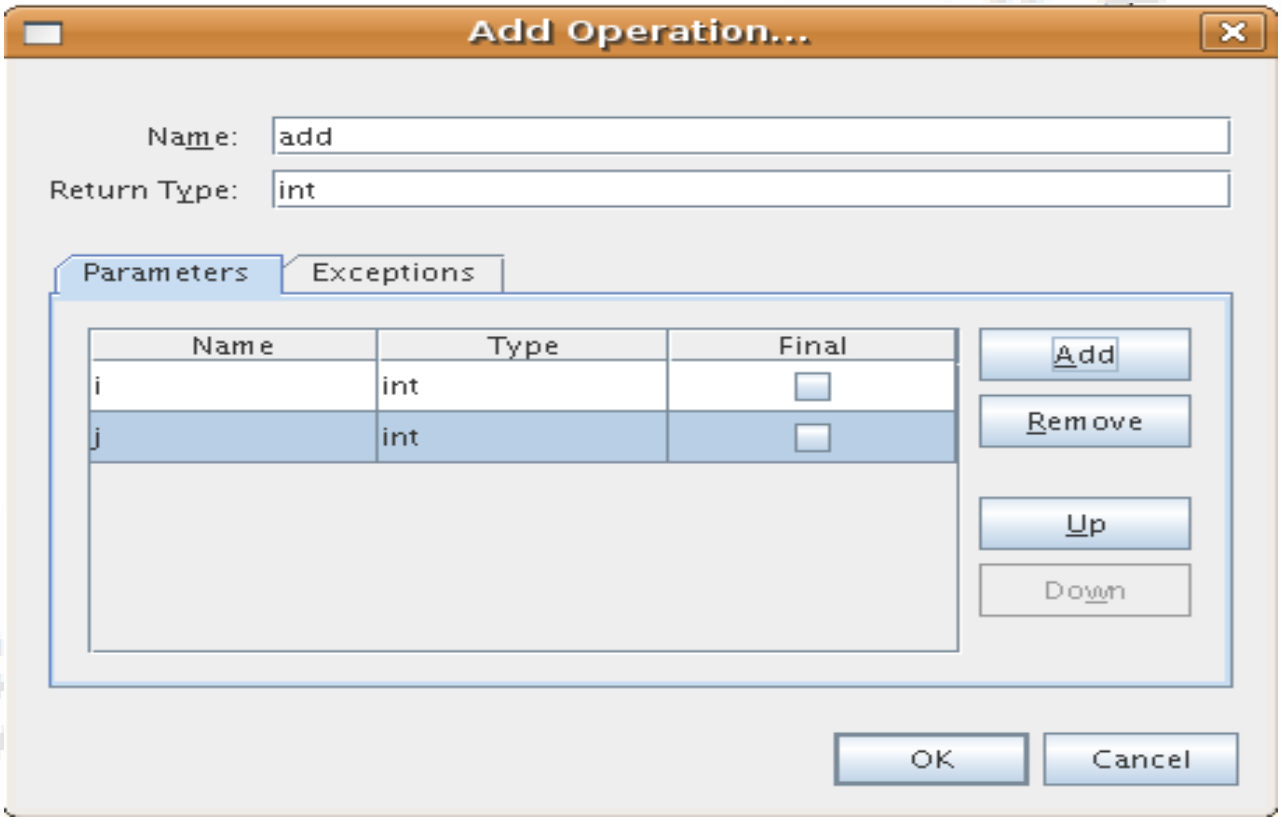
WEB-SERVICES IMPLEMENTATION

- **Step 3: Adding an Operation to the Web Service :**
- Find the web service's node in the Projects window. Right-click that node. A context menu opens.
- Click **Add Operation** in either the visual designer or the context menu. The Add Operation dialog opens.



WEB-SERVICES IMPLEMENTATION

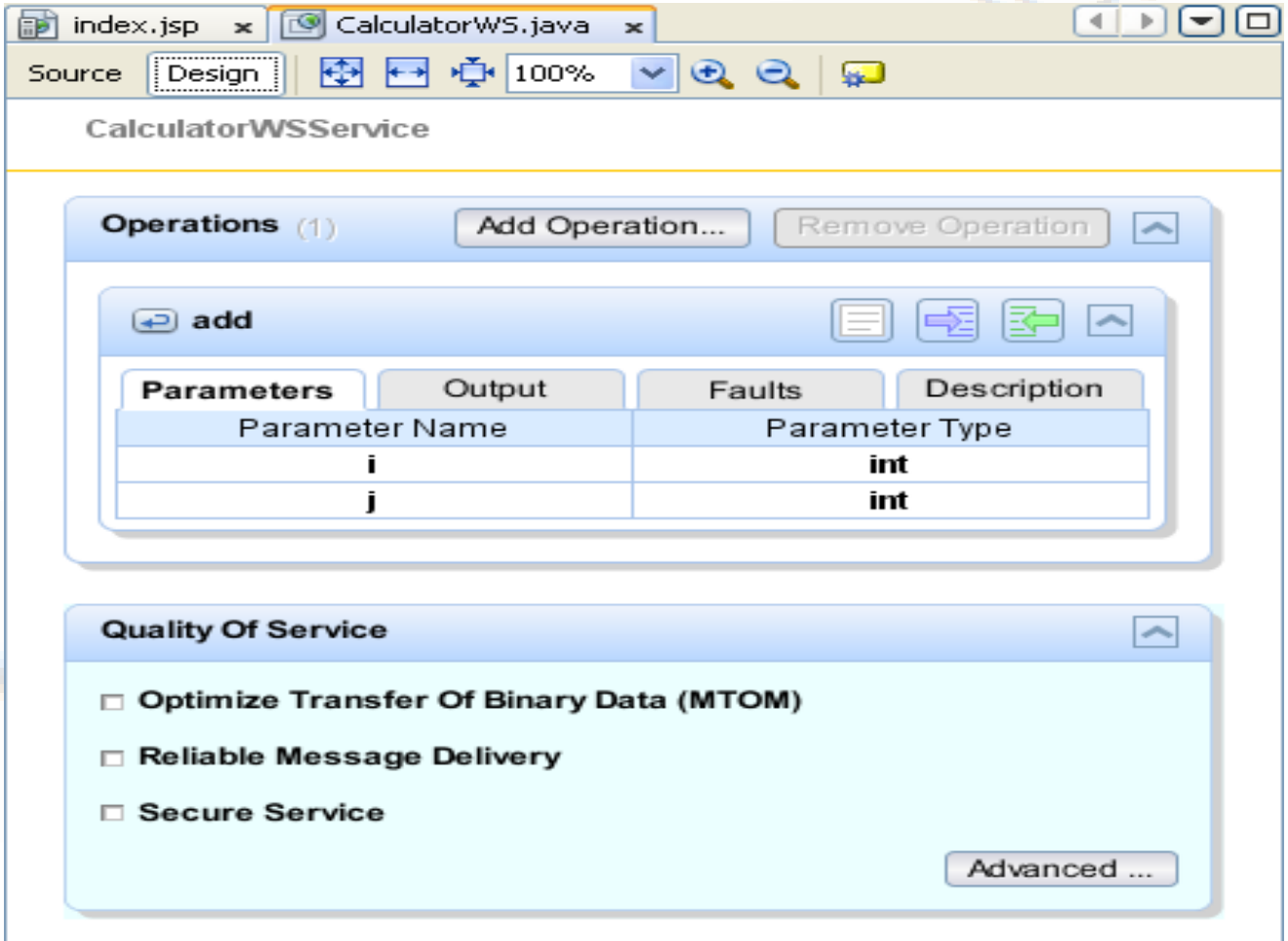
- In the upper part of the **Add Operation dialog box**, type **`add`** in **Name** and type **`int`** in the **Return Type** drop-down list.
- In the lower part of the Add Operation dialog box, click **Add** and create a parameter of type **`int`** named **`i`**. Click **Add** again and create a parameter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



- Click **OK** at the bottom of the Add Operation dialog box. You return to the editor.

WEB-SERVICES IMPLEMENTATION

- 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 looks like as below:



WEB-SERVICES IMPLEMENTATION

- Click **Source** menu and can view the generated code as follows:

```
package org.me.calcuator;

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;
    }
}
```

WEB-SERVICES IMPLEMENTATION

- 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 above code, the web service simply receives two numbers and then returns their sum.

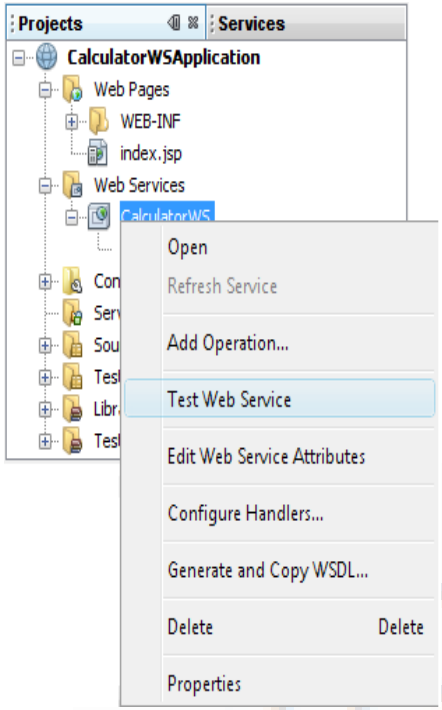
WEB-SERVICES IMPLEMENTATION

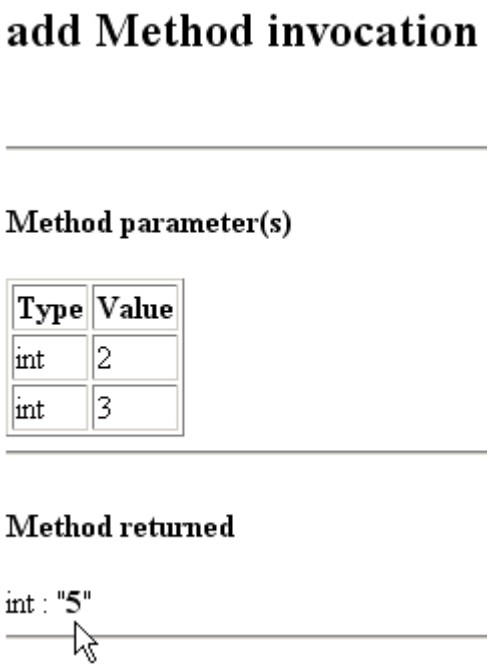
- **Step 4: Deploying and Testing the Web Service :** Once you deploy a web service to a server, you can use the IDE to open the server's test client. The GlassFish server provide test clients whereas in Tomcat Web Server, there is no test client.

- Right-click the **project** and choose **Deploy**. The IDE starts the application server, builds the application, and deploys the application to the server.
- 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.**

WEB-SERVICES IMPLEMENTATION

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





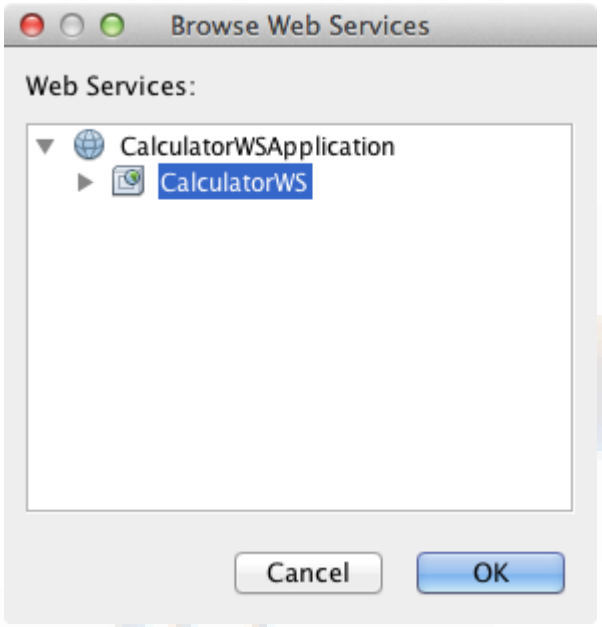
WEB-SERVICES IMPLEMENTATION

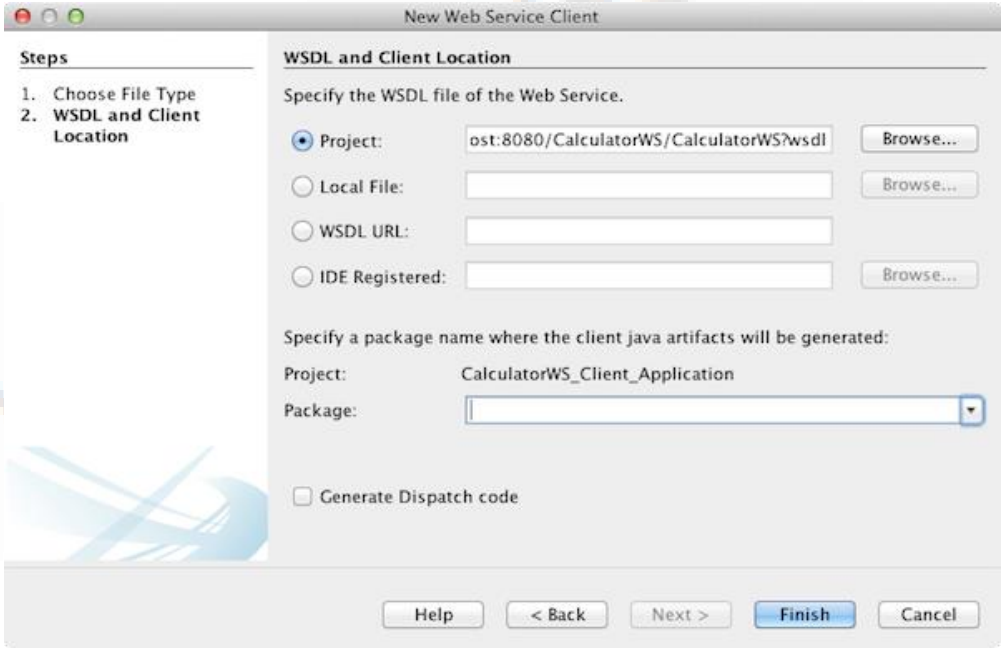
- **Step 5: Consuming the Web Service :** Once the web service is deployed, you need to **create a client** to make use of the web service's **add** method. Here, you can create three types of clients : a Java class in a Java SE application, a servlet, and a JSP page in a web application.

- Client 1: Java Class in Java SE Application
- Choose **File** > **New Project**.
- Select **Java Application** from the **Java** category.
- Name the project `CalculatorWS_Client_Application`.
- Keep "**Create Main Class selected**" and accept all other default settings. Click **Finish**.
- Right-click the `CalculatorWS_Client_Application` node and choose **New** > **Web Service Client**. The New Web Service Client wizard opens.

WEB-SERVICES IMPLEMENTATION

- Select “Project as the **WSDL…** source. Click on **Browse** button. Browse to the CalculatorWS web service in the **CalculatorWSApplication** project. When you have selected the web service, click **OK**.

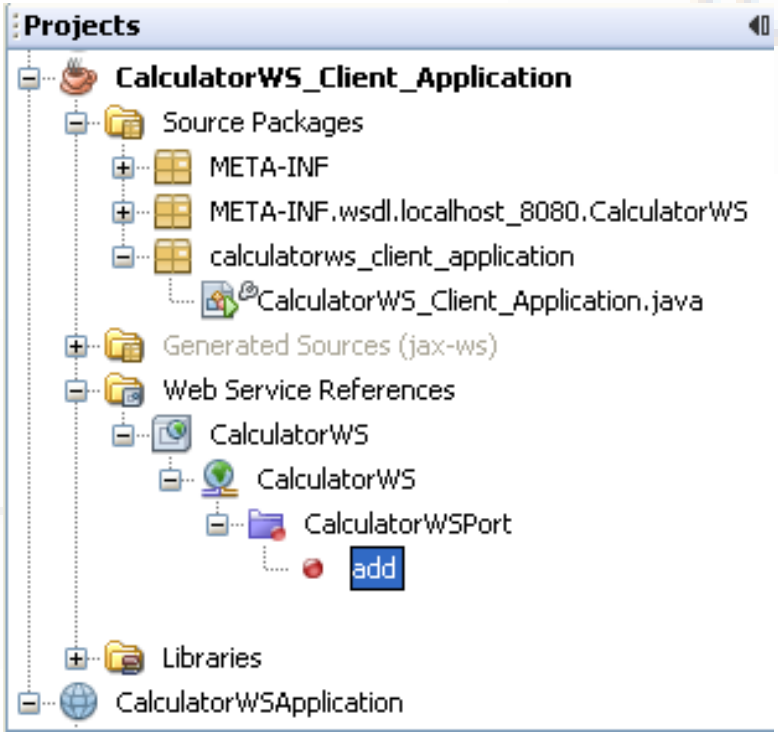




- Do not select **a package** name. Leave this field empty. Keep the other settings at default and click **Finish**.

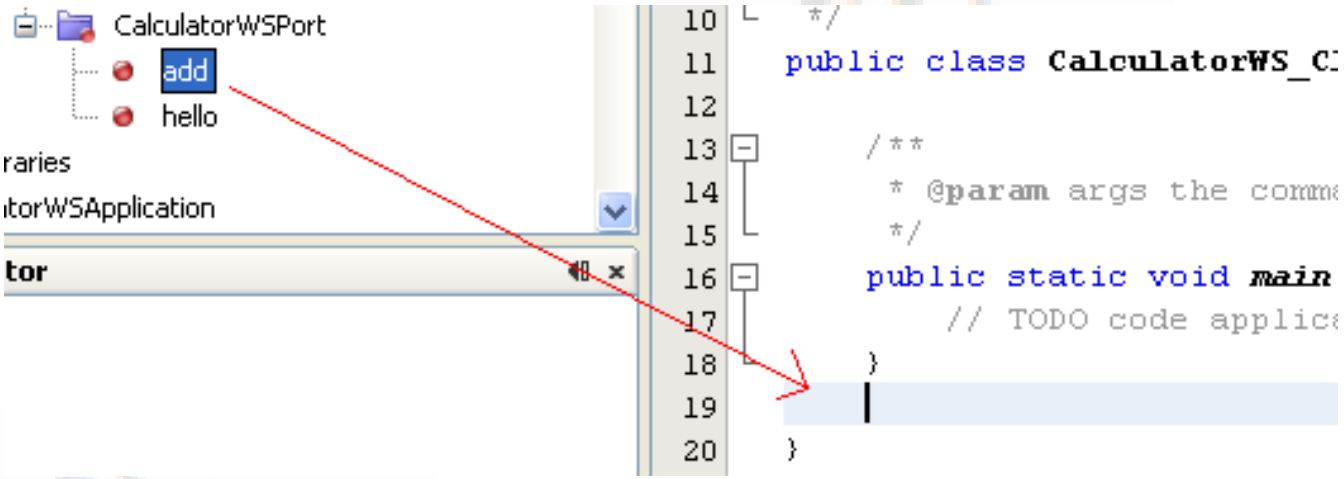
WEB-SERVICES IMPLEMENTATION

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



WEB-SERVICES IMPLEMENTATION

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



WEB-SERVICES IMPLEMENTATION

Now you can see the code as per below:

```
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);
}
```

WEB-SERVICES IMPLEMENTATION

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

```
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);
    }
}
```

Compile and Execute

- 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)
```

References

- Netbeans IDE with glassfish server : https://netbeans.org/downloads/
- Getting Started with JAX-WS Web Services : https://netbeans.org/kb/docs/websvc/jax-ws.html
- Getting Started with RESTful Web Services: https://netbeans.org/kb/docs/websvc/rest.html