

**Module code and title: 5COSC010C -Client Service Architecture
Tutorial Manual**

Tutorial title	User-Defined Types and Exceptions in Web Services
Tutorial type	Guided and independent and non-marked
Week 05	13/02/2020

Contents

Learning Goals	1
TASKS to be Performed under the instruction of the Tutor (from Task 1 to Task 19)	1
TASKS to BE PERFORMED Independently by the student (from Task 20 to Task 23) (Formative Assessment).....	13

Learning Goals

This tutorial focuses on two main learning goals:

- To apply the use of User-Defined types, and,
- Exceptions in Web Services
- Solution for passing arrays (need to encapsulate inside another class)

It is divided into two separate sections, the student will perform the first task (1-25) following the instructions of the tutor, and then, will complete the other tasks (26 to 31) independently.

TASKS to be Performed under the instruction of the Tutor (from Task 1 to Task 25)

- 1) Start Netbeans in your system.
- 2) Create a new java project in Netbeans (Figure 1 to 3). We will call this project **TutorialFour** and it will consist in the Web Service implementation of Tutorial 2 – Dummy Project 2 – Types and Exceptions.

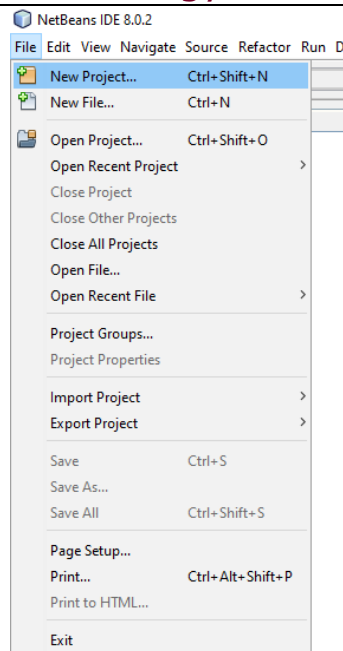


Figure 1, Create a Web Application Project in NetBeans (1)

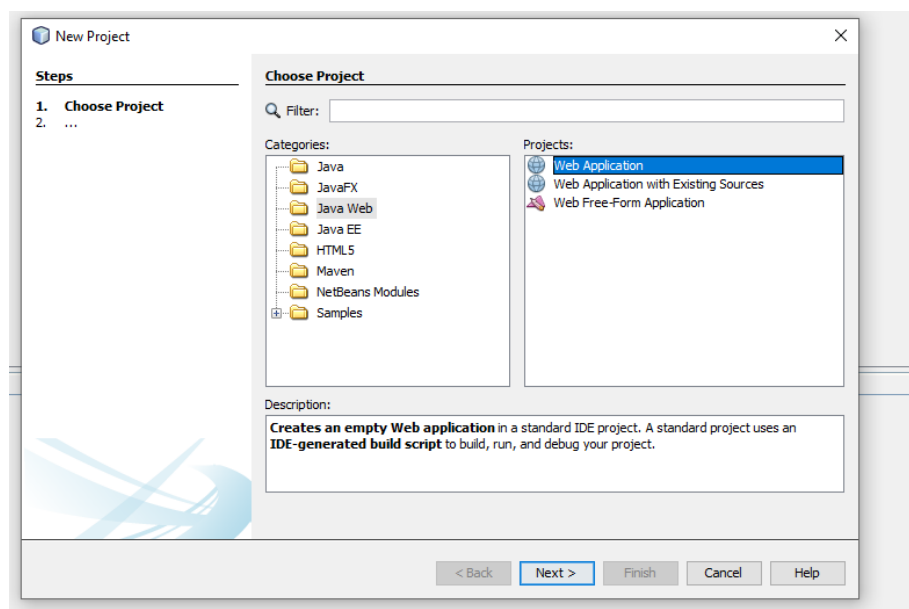


Figure 2, Create a Web Application Project in NetBeans (2)

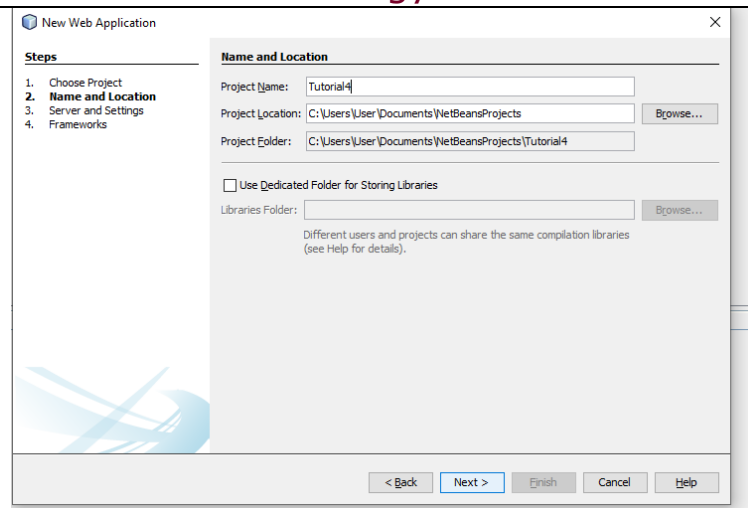


Figure 3, a Web Application Project in NetBeans (3)

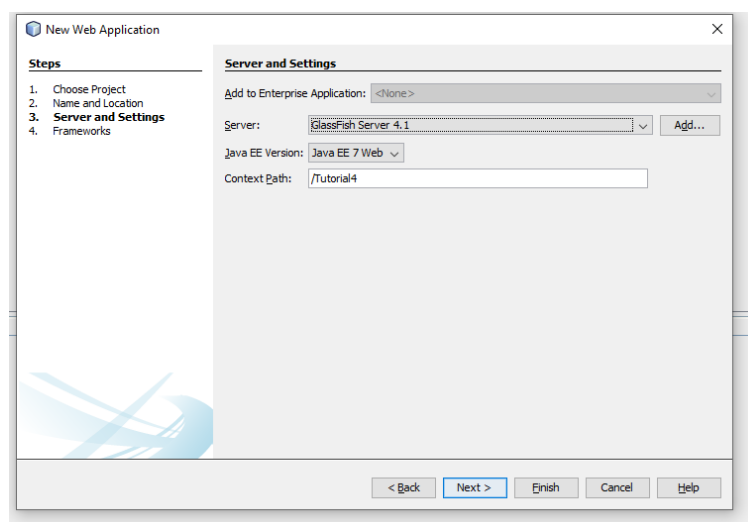


Figure 4, Create a Web Application Project in NetBeans (4),

Select GlassFish Server

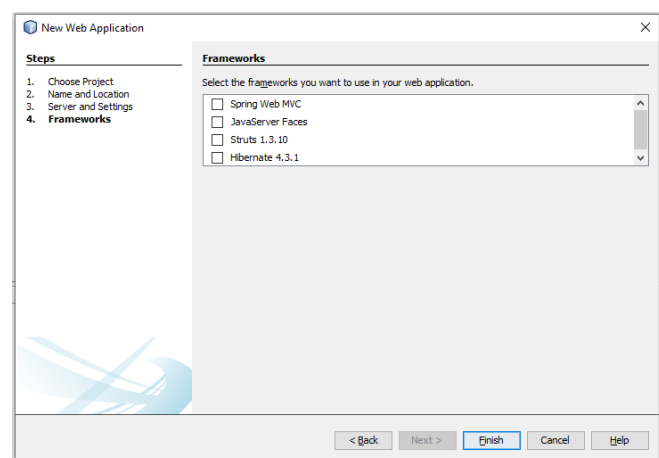


Figure 5, Create a Web Application Project in NetBeans (5) - Do not select any framework

- 3) Create a new Java package in the Web Application where we will put our server, called it server (Figure 6, Creating a server Java Package).

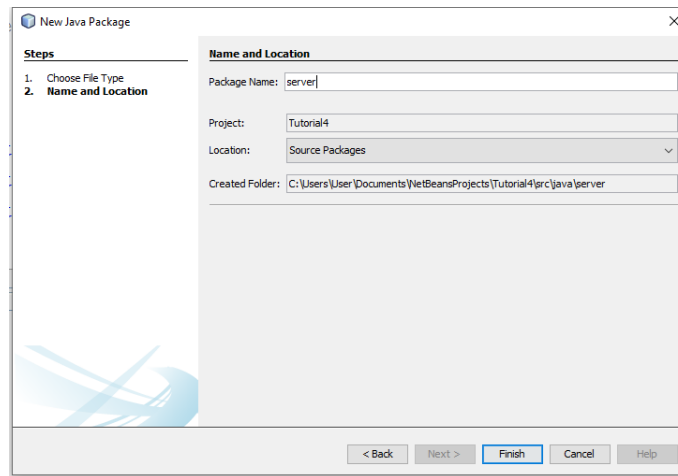


Figure 6, Creating a server Java Package

- 4) Now, we can create a proper Web Service, we will call it Tutorial4WebService

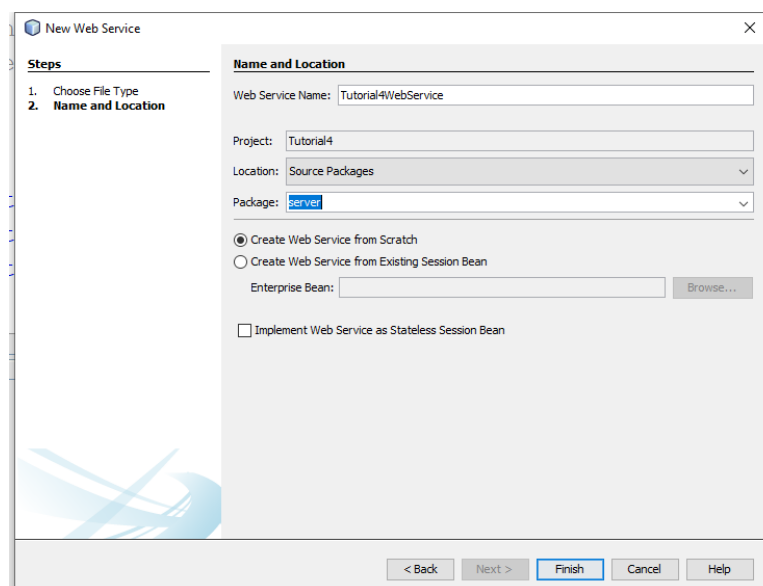


Figure 7, Create the Web Service

- 5) You can observe that NetBeans creates a standard method called Hello which returns (as a String) the message sent (passed as a String). (Figure 8, Standard Web Service Created by Netbeans.)

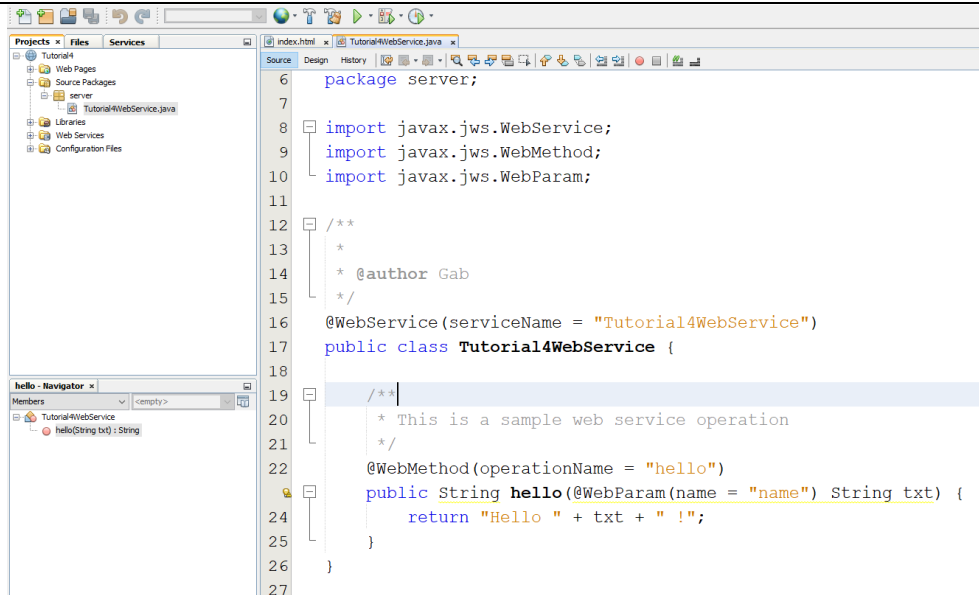


Figure 8, Standard Web Service Created by Netbeans.

- 6) We can use the design view of the Web Service to add and remove methods, you can see the hello method, its parameters and the return type. This method is added automatically by the Web Services creator.

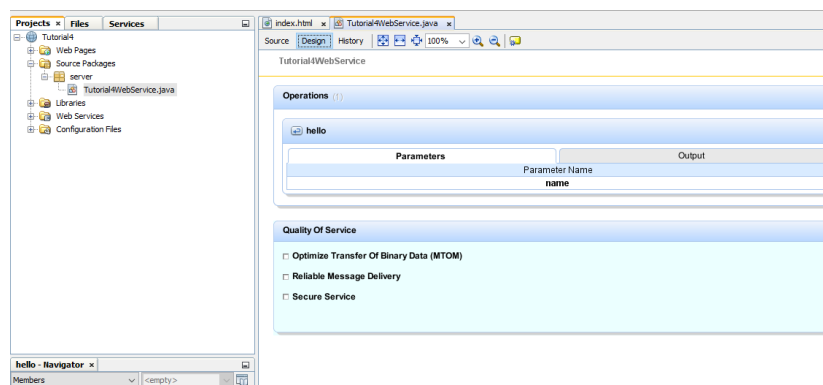


Figure 9, Design view of the Web Service

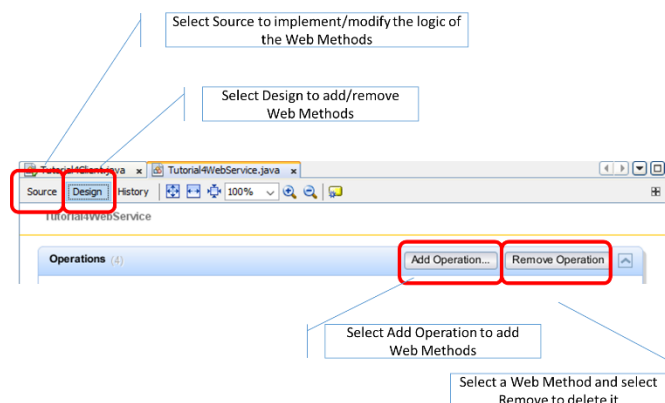


Figure 10, Details of the functionalities of the Web Services Interface.

- 7) We use the AddOperation Button to add our usual isConnected method that returns true if the server is connected (Figure 11, Adding the isConnected method to the Serve). Netbeans will create an empty method which you have to complete in the next step.

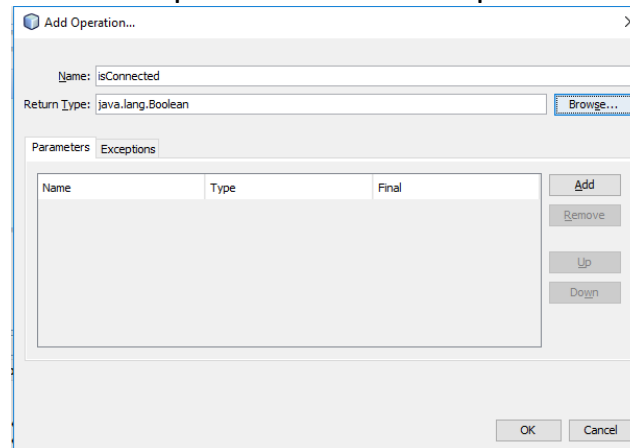


Figure 11, Adding the isConnected method to the Server

- 8) Add the usual simple logic of the usual isConnected method in the logic of the Web Service (Figure 12, Logic on the Web Service Method isConnected.)

```
/**
 * Web service operation
 */
@WebMethod(operationName = "isConnected")
public Boolean isConnected() {
    //TODO write your implementation code here:
    System.out.println("[SERVER] - Testing Connection...");
    return true;
}
```

Figure 12, Logic on the Web Service Method isConnected.

- 9) Now we deploy the server on the Glassfish server engine. **ALWAYS REMEMBER TO REDEPLOY THE SERVER EVERY TIME YOU CHANGE ITS CODE.**
- 10) NetBeans offers us a testing tool without even to have to write a client (NetBeans will write a simple client in a browser)(Figure 13, Testing the Web Service). Test the isConnected method, it returns true, that is correct! You can also check the server log to see if it is correct.

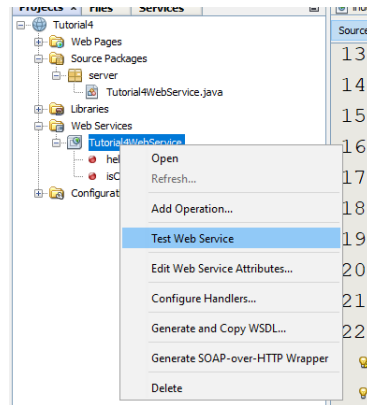


Figure 13, Testing the Web Service

- 11) Now we create a client, this is going to be a separate Java Application Project (NOT A WEB APPLICATION, A SIMPLE JAVA APPLICATION) (Figure 14, Creating the Client (Step 1) and Figure 15, Creating the Client (Step 2))

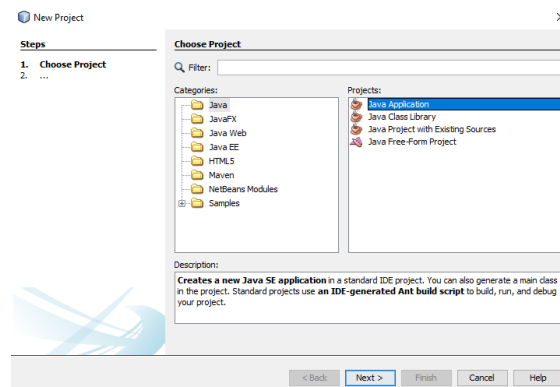


Figure 14, Creating the Client (Step 1)

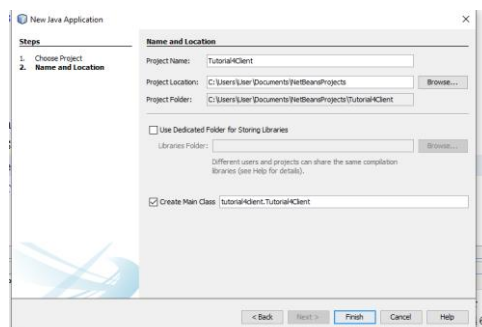


Figure 15, Creating the Client (Step 2)

Obtaining the hosted Webservice Description Language File (**WSDL**) to generate "**Client Stub**". This file is dynamically generated by supporting libraries in GlassFish. Else you need to write it by hand ! (**Server Stubs** are managed by Glassfish SOA libraries).

- 12) Now we build a communication system between the client and the server. In order to do that, we have to create what is called a client stub (or Web Service Client) on the client which will be able to communicate with the Server. NetBeans will create behind the scenes all the code that handles the communication, no horrible XML coding by hand ! Select the Web Service under Project and leave the package empty (Figure 16, Creating the Web Service Client (Step 1), and, Figure 17, Creating the Web Service Client (Step 2)).

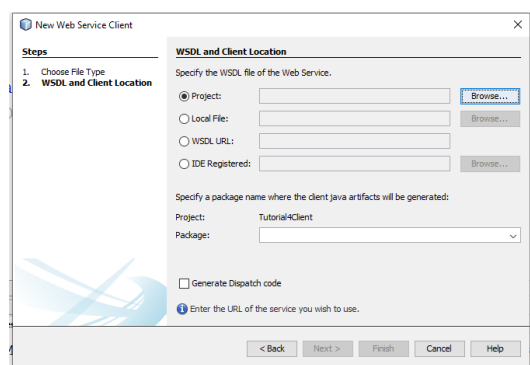


Figure 16, Creating the Web Service Client (Step 1)

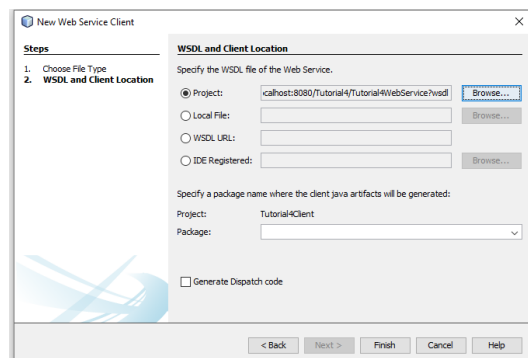
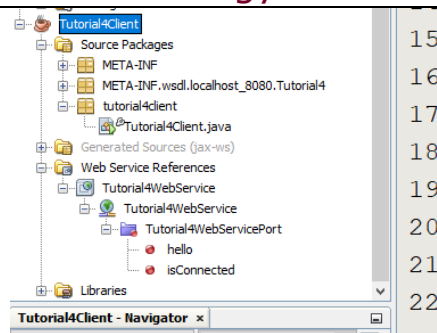


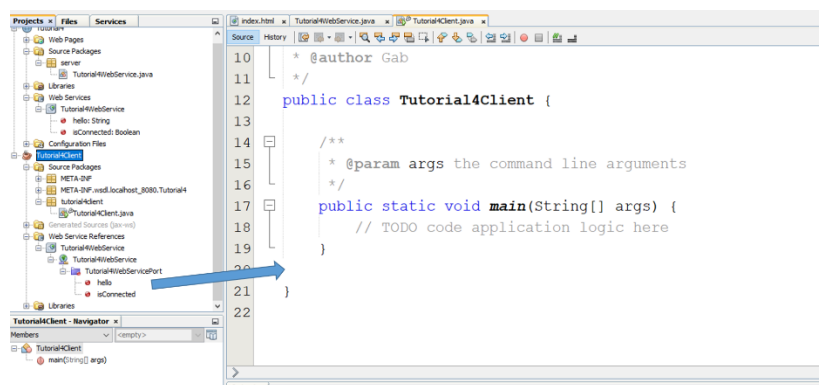
Figure 17, Creating the Web Service Client (Step 2)

- 13) You can notice that now on the client side, we have a representation of the server web methods. These are called stubs and they are an interface to the network that will handle the communication with the server (Figure 18, Client Stubs).

Challenge : Try to figure out which client stub class has the Service Producers "**End Point**" address.



- 14) If you want to use one of the remote methods on the server, you just have to drag and drop the icon of the method in the client code where you want to use it, this will create a client stub: a method on the client capable of connecting to the internet to connect to a server (Figure 19, Creating Client Stubs).



- 15) Now you can invoke your client stub (which will in turn connect to the network and call the server) from your code (Figure 20, Invoking the client stub).

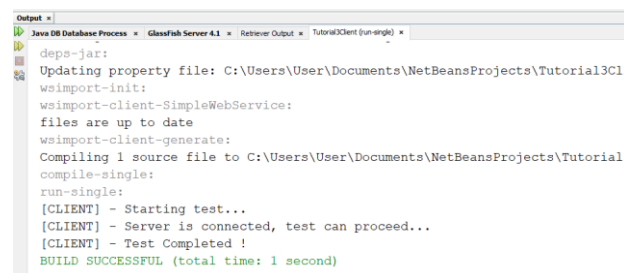
```
private void execute()
{
    System.out.println("[CLIENT] - Starting test...");
    if (isConnected())
    {
        System.out.println("[CLIENT] - Server is connected, test can proceed...");
    }
    else
    {
        System.out.println("[CLIENT] - Server is NOT connected, test failed.");
    }
    System.out.println("[CLIENT] - Test Completed!");
}
```

Figure 20, Invoking the client stub

- 16) **ONLY If the client code does not compile because it cannot find the classes in the server package of the client stubs, you can find the**

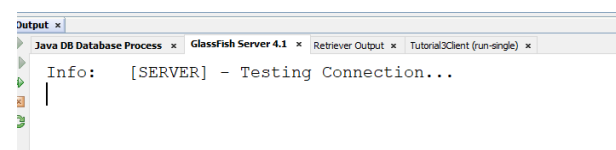
code under Generated Sources (jax-ws). Simply copy and paste the entire java package server into your client code.

- 17) Now we are ready to run the client and see if it is really capable of connecting to a server running in a separate project.
- 18) Observe how the logging calls (system.out.println(...)) from the Server and the Client appear in different stdout streams (one from the GlassFish Server and one from the Client).



```
Output:
Java DB Database Process x GlassFish Server 4.1 x Retriever Output x Tutorial3Client (run-single) x
deps-jar:
Updating property file: C:\Users\User\Documents\NetBeansProjects\Tutorial3Cl1
wsimport-init:
wsimport-client-SimpleWebService:
files are up to date
wsimport-client-generate:
Compiling 1 source file to C:\Users\User\Documents\NetBeansProjects\Tutorial3
compile-single:
run-single:
[CLIENT] - Starting test...
[CLIENT] - Server is connected, test can proceed...
[CLIENT] - Test Completed !
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 21, Client Log



```
Output:
Java DB Database Process x GlassFish Server 4.1 x Retriever Output x Tutorial3Client (run-single) x
Info: [SERVER] - Testing Connection...
|
```

Figure 22, Server Log

Adding methods to the “Service Provider”

- 19) Add a method to the server that adds two Double numbers (Figure 23, add the add method to the Web Service) with some simple logging information (Figure 24, implement the add method to the Web Service).

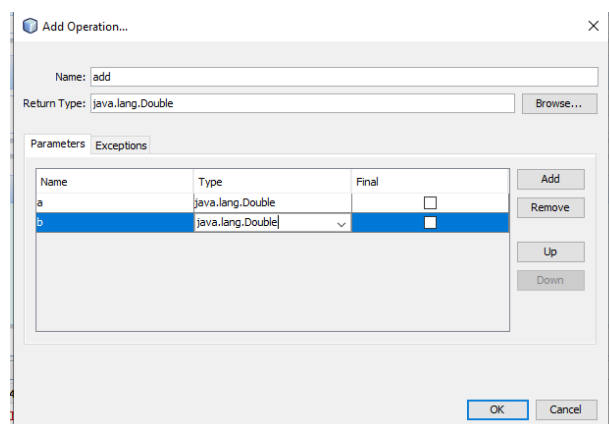


Figure 23, add the add method to the Web Service

```
/**
 * Web service operation
 */
@WebMethod(operationName = "add")
public Double add(@WebParam(name = "a") Double a, @WebParam(name = "b") Double b) {
    System.out.println("[SERVER] - add(" + a + ", " + b + ")");
    return a+b;
}
```

Figure 24, implement the add method to the Web Service

- 20) Unit test the new web method with the “Test Web Service” function (you did this last time)
- 21) Develop the client side code and update the client stubs to reflect the changes on the server, **REMEMBER TO RE-DEPLOY THE SERVER.**

```
{
    System.out.println("[CLIENT] - Starting Test...");
    if (isConnected())
    {
        System.out.println("[CLIENT] - Server is connected, continuing test...");
        Double x = 2.0;
        Double y = 3.1;
        Double z = add(x, y);
        System.out.println("[CLIENT] - The Server has returned: " + z + " = " + x + " + " + y);
    }
    else

```

Figure 25, Implement the Client

- 22) Test the Client.

Challenge: What type of **IPC** was used here ?

Adding Validations with Exceptions on the **Service Provider**

- 23) We now want to take into account the fact that we cannot add a number to null. Hence we will create a new method (addWithExceptions) with faults (the Web Service equivalent of an Exception) on our Web Service. (Figure 26, Creating a Web Method with Faults (Exceptions), Step A, Figure 27, Creating a Web Method with Faults (Exceptions), Step B, Figure 28, Creating a Web Method with Faults (Exceptions), Step C). **BE EXTRA CAREFUL IN SELECTING the java.lang.Exception type !**

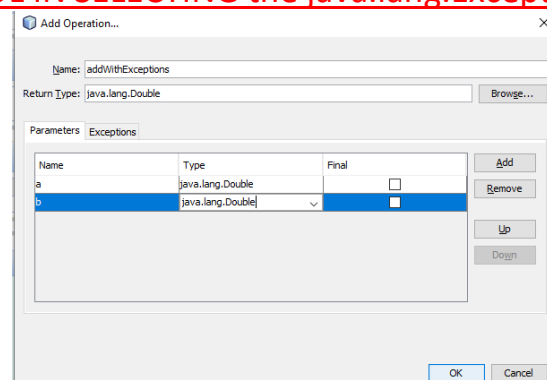


Figure 26, Creating a Web Method with Faults (Exceptions), Step A

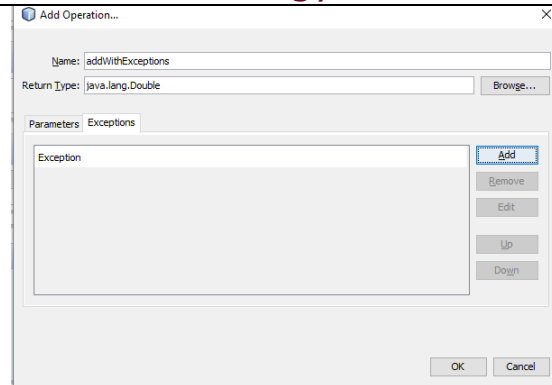


Figure 27, Creating a Web Method with Faults (Exceptions), Step B

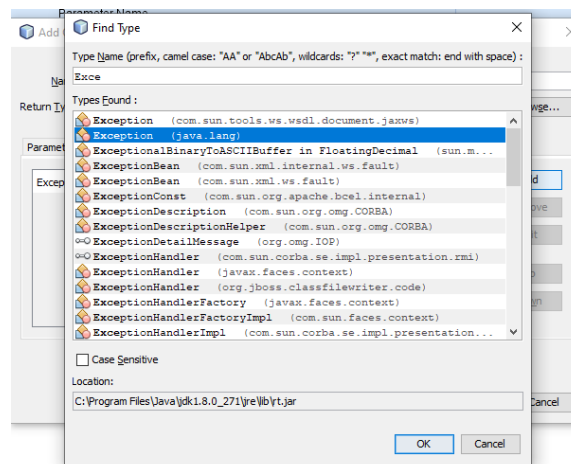


Figure 28, Creating a Web Method with Faults (Exceptions), Step C

Adding code to Anticipate Exceptions on the **Service Consumer**

- 24) Develop the client (updating the client stubs) to test the exception method with all combinations of parameter: (value, value), (null, value), (value, null), and, (null, null) (Figure 29, Catching and Testing Exceptions on the Client Side). You can refer to Tutorial 2 for a detailed description on how to throw and catch Exceptions in Java.

```
x = 2.0;
y = 3.1;
try {
    z = addWithException(x, y);
    System.out.println("[CLIENT] - The Server has returned: " + z + " = " + x + " + " + y);
} catch (Exception_Exception ex) {
    Logger.getLogger(Tutorial4Client.class.getName()).log(Level.SEVERE, null, ex);
}

x = null;
y = 3.1;
try {
    z = addWithException(x, y);
    System.out.println("[CLIENT] - The Server has returned: " + z + " = " + x + " + " + y);
} catch (Exception_Exception ex) {
    Logger.getLogger(Tutorial4Client.class.getName()).log(Level.SEVERE, null, ex);
}

x = 2.0;
y = null;
try {
    z = addWithException(x, y);
    System.out.println("[CLIENT] - The Server has returned: " + z + " = " + x + " + " + y);
} catch (Exception_Exception ex) {
    Logger.getLogger(Tutorial4Client.class.getName()).log(Level.SEVERE, null, ex);
}

x = null;
y = null;
try {
    z = addWithException(x, y);
    System.out.println("[CLIENT] - The Server has returned: " + z + " = " + x + " + " + y);
}
```

Figure 29, Catching and Testing Exceptions on the Client Side

- 25) Finally, export both the projects (client and server) as zip file on Netbeans. Upload to the tutorial link.

Challenge: How many tests should you perform for the addWithException method to be confident that all possible options are tested ?

TASKS to BE PERFORMED Independently by the student (from Task 26 to Task 31) (Formative Assessment)

- 26) Modify the server by adding methods that subtract, divide and multiply numbers with and without Exceptions.
- 27) Test these new method from the client
- 28) What happens if you divide a number with zero like 4 and 0 ? Take this into account into the divideWithException Web Method. How can you solve it ?

Challenge: Current libraries with Netbeans does not support Web Methods with Array parametes. What can you propose if you want to “pack” an array to be sent to the server and vice versa ? (Hint: Encapsulation)

- 29) Modify the server by adding methods that perform the four operations on arrays and not juts simple numbers. As an example $[1,2,3] + [3,4,5] = [4,6,8]$.
- 30) Think on the exceptions that these methods need to raise, what if you try to perform an operation of arrays of different sizes ? As an example $[1,2,3] + [3,4]$, what would you do ? Raise an exception ? If you decide so, add the code that throws the exception in the server and catches it in the client.
- 31) Think on the exceptions that these methods need to raise, what if you try to perform an division of arrays and only some numbers are not valid ? As an example $[1,2,3,0] / [3,0,4,0]$, what would you do ? Raise an exception and return no results (even for the valid numbers) ? Add the code that throws the exception in the server and catches it in the client.