

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 06	18/02/2020

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).....	17

Learning Goals

This tutorial focuses on two main learning goals:

- To implement correlated and stateful Web Methods on a Web Service.
- To use iterations on collections.

It is divided into two separate sections, the student will perform the first task (1-27) following the instructions of the tutor, and then, will complete the other tasks (28 to 31) independently.

TASKS to be Performed under the instruction of the Tutor (from Task 1 to Task 27)

- 1) Start Netbeans (8.0.2 EE or *.2 EE) in your system.
- 2) Create a new Web Application project in Netbeans. We will call this project ***Tutorial5***.

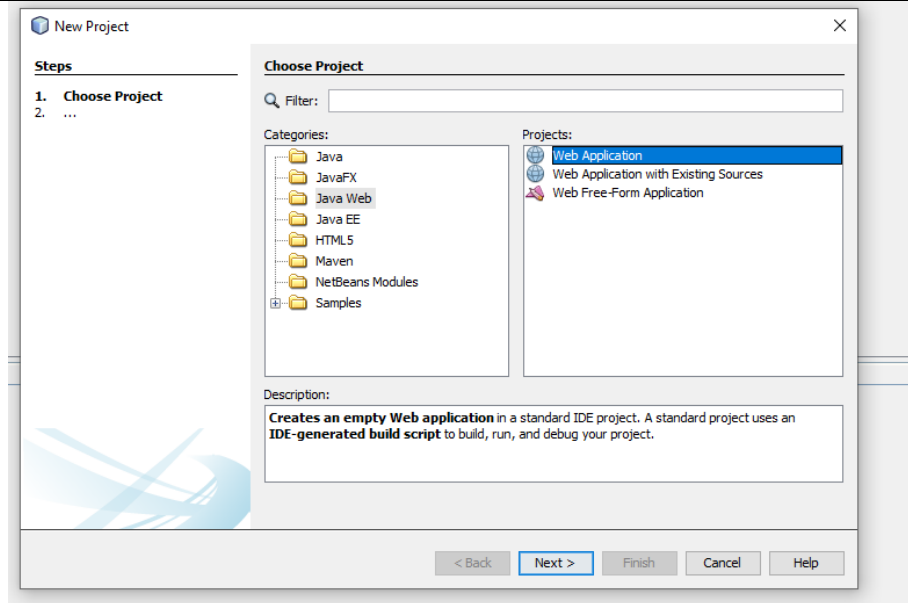
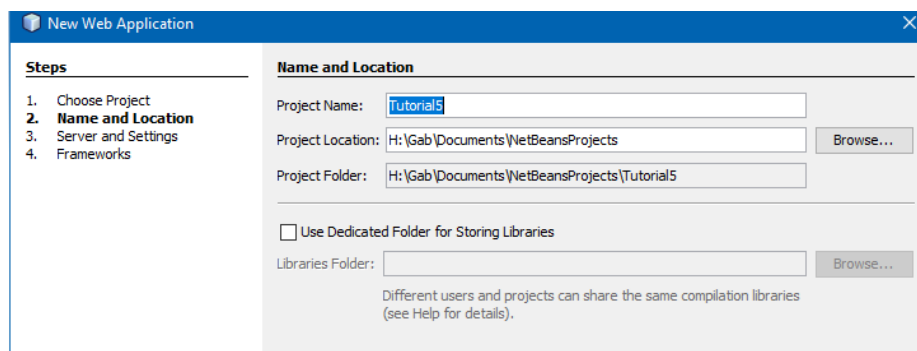


Figure 1, Create a Web Application Project in NetBeans (2)



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

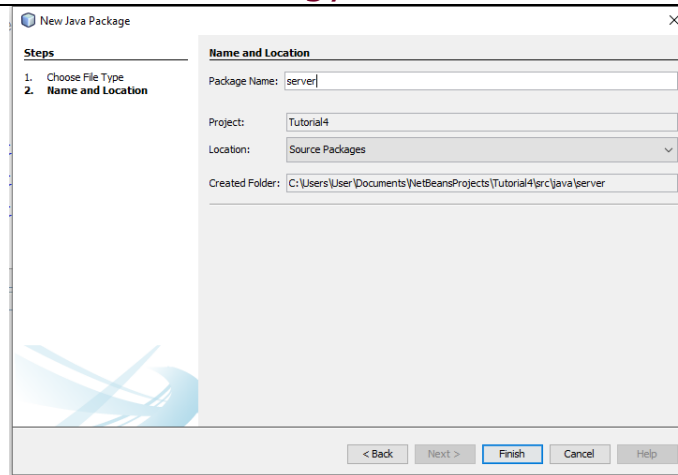


Figure 3, Creating a server Java Package

- 4) Now, we can create a proper Web Service, we will call it Tutorial5WebService ()

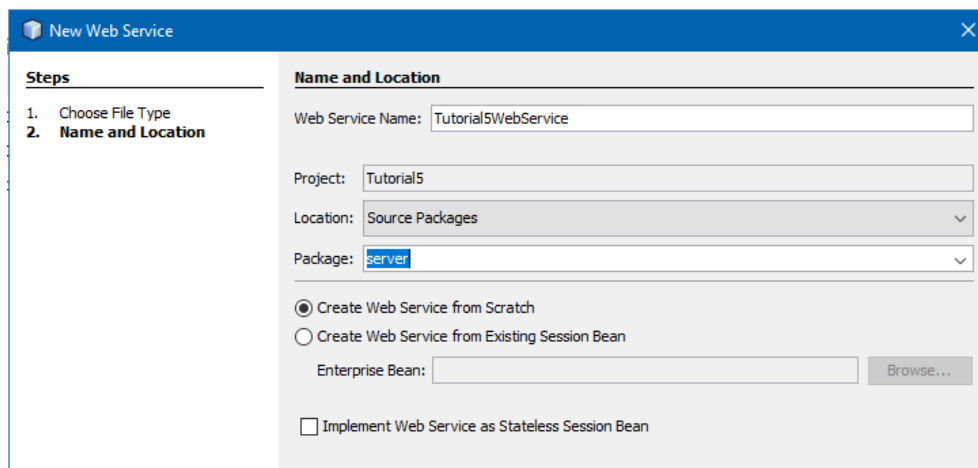


Figure 4, Create the Web Service

- 5) You can observe that NetBeans creates a standard method called Hello which returns (as a String) the message sent (passed as a String).
- 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.

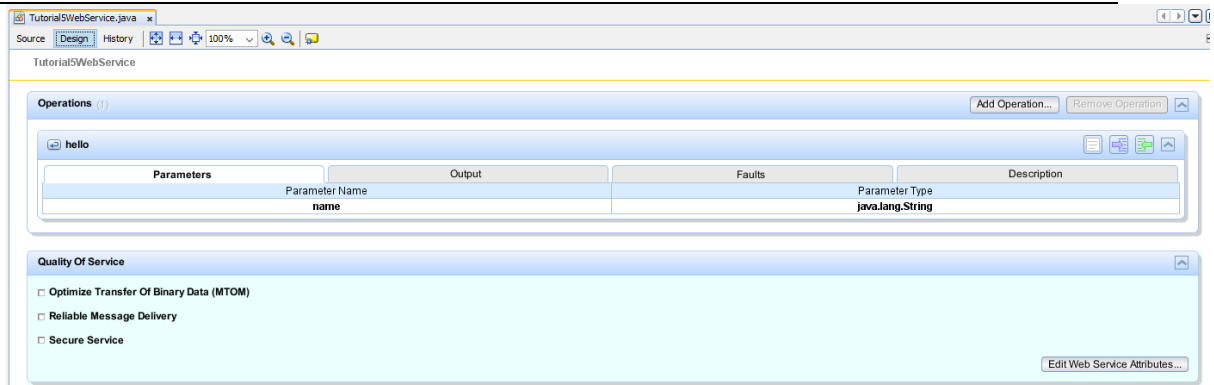


Figure 5, Design view of the Web Service

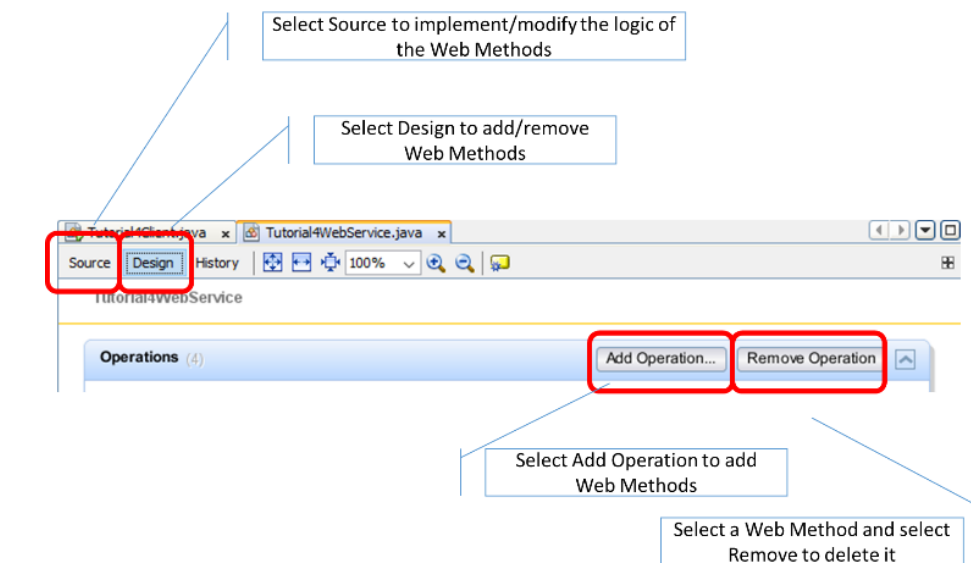


Figure 6, Details of the functionalities of the Web Services Interface.

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

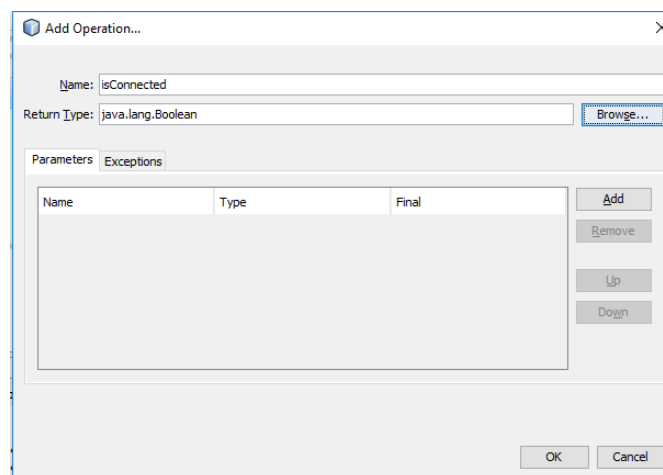


Figure 7, Adding the isConnected method to the Server

- 8) Add the usual simple logic of the usual isConnected method in the logic of the Web Service (Figure 8, 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 8, 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 9, 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.

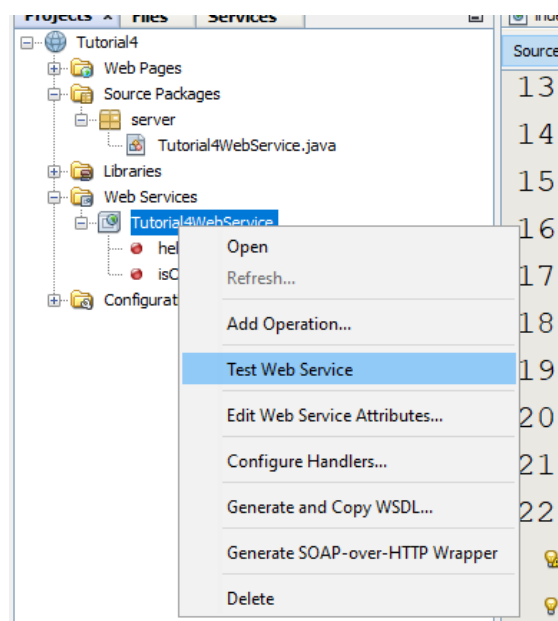


Figure 9, Testing the Web Service

UNIT TEST IS OPTIONAL = sometimes it does not work. You can append "?Tester" to the URL instead of "?wsdl" to see if the tester pops up. If not ... simply skip this step.

- 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 10, Creating the Client (Step 1) and Figure 11, Creating the Client (Step 2))

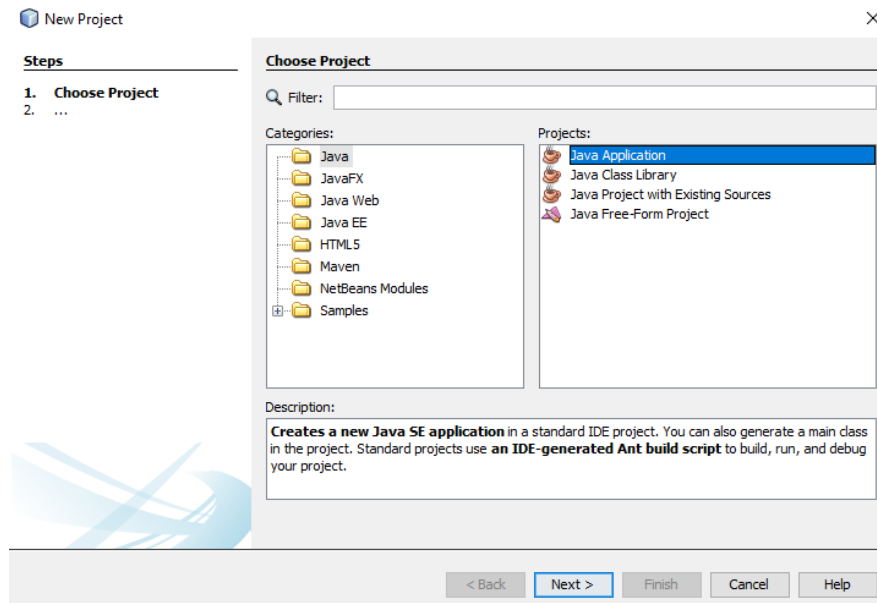


Figure 10, Creating the Client (Step 1)

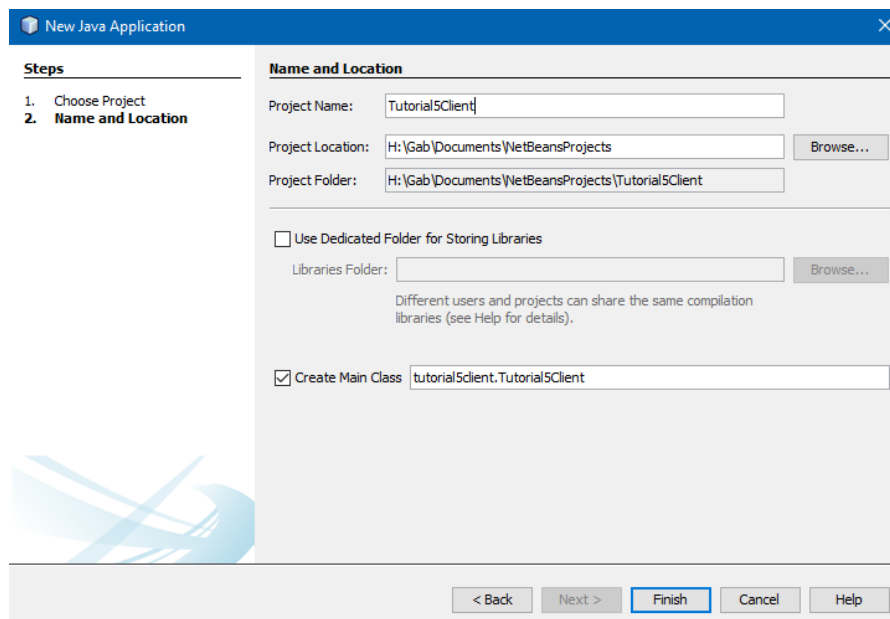


Figure 11, Creating the Client (Step 2)

- 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. Select the Web Service under Project and

leave the package empty (Figure 12, Creating the Web Service Client (Step 1)), and, Figure 13, Creating the Web Service Client (Step 2)).

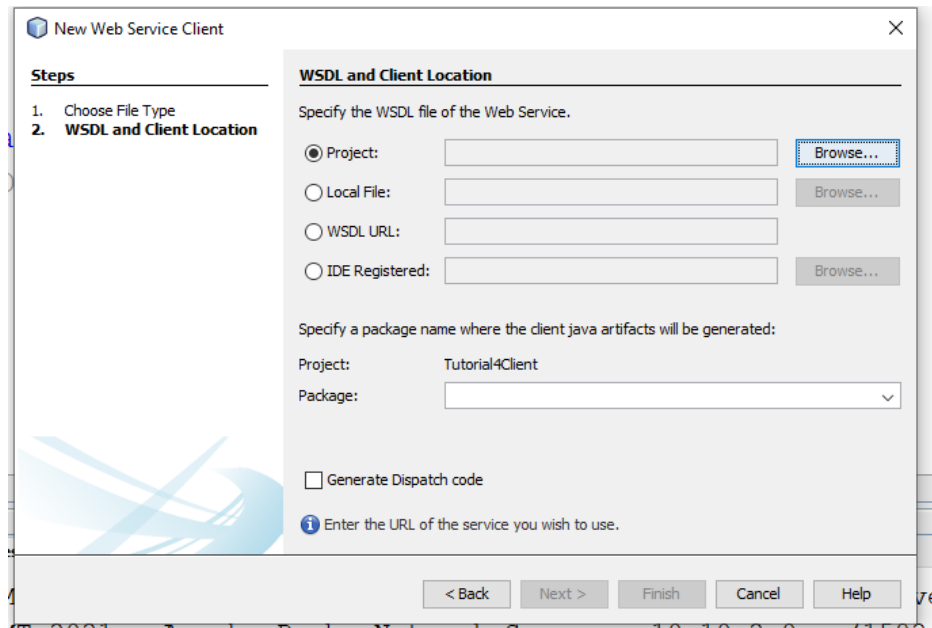


Figure 12, Creating the Web Service Client (Step 1)

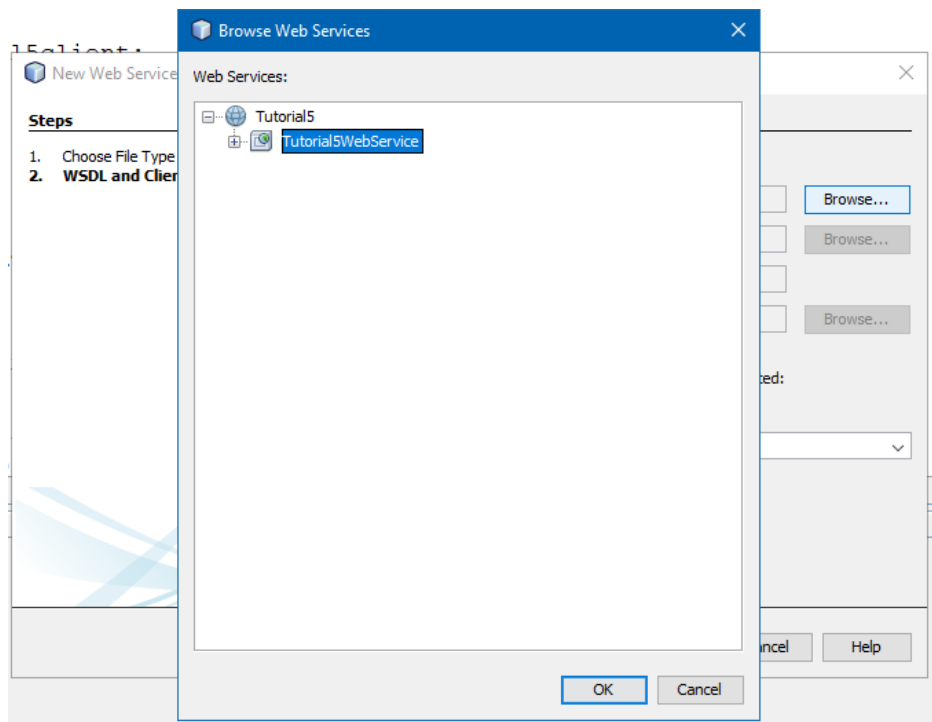


Figure 13, Creating the Web Service Client (Step 2)

Netbeans actually “scrapes” the WSDL = if you look closely at the “Project” box after selection, you can see that Netbeans actually appends “?wsdl” to your URL. This means you can scrape the file your self and use the “Local File” option to feed a offline copy of the wsdl file.

- 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 14, Client Stubs).

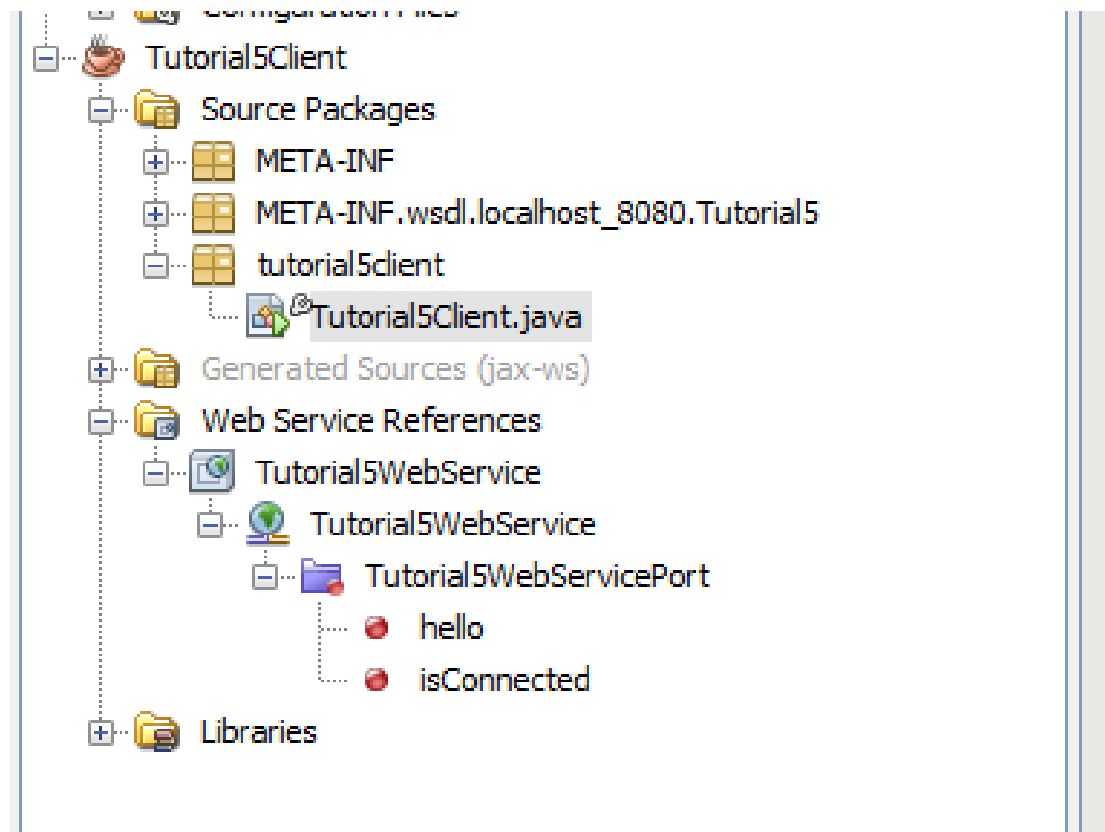


Figure 14, Client Stubs

- 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 15, Creating Client Stubs).

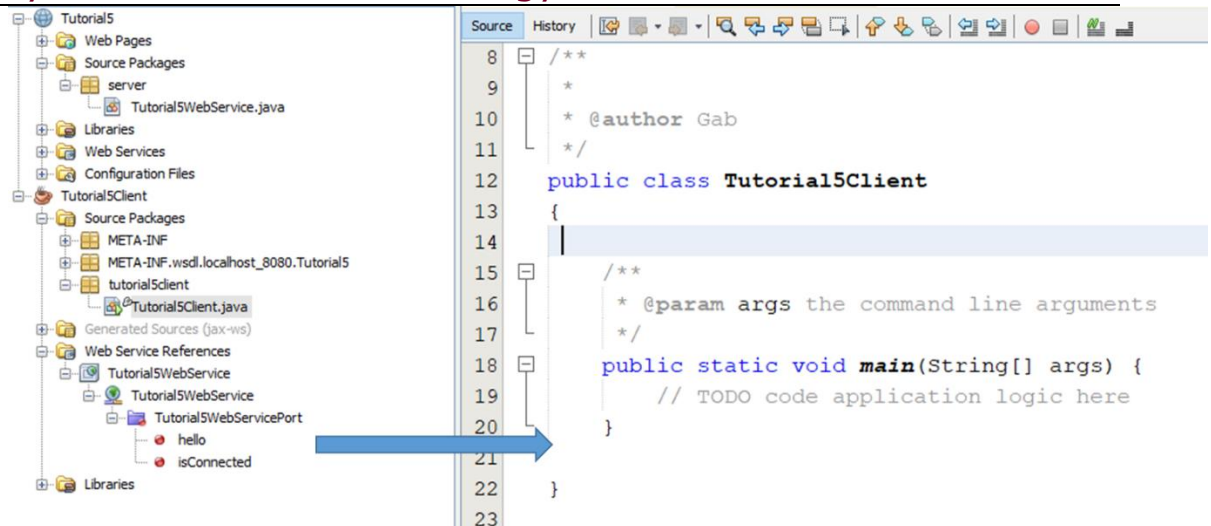


Figure 15, Creating Client Stubs

- 15) Now you can invoke your client stub (which will in turn connect to the network and call the server) from your code (**Error! Reference source not found.**). Develop the Client-side code by creating an instance of Tutorial5Client and invoking a execute method. Develop the execute method to connect to the remote server (Figure 16, Client code)

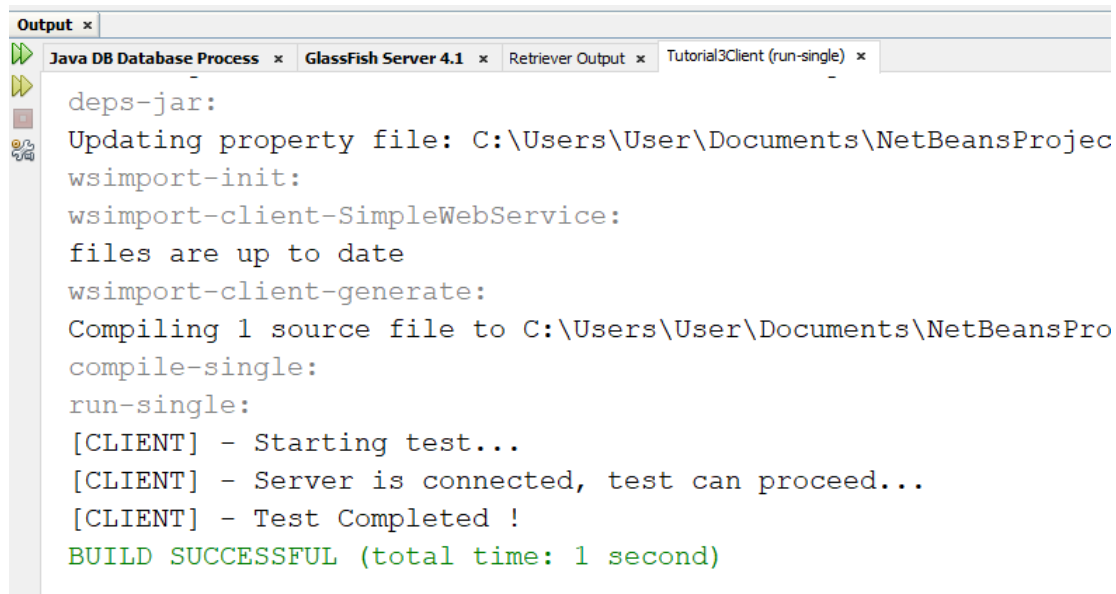
```
1 public static void main(String[] args) {
2     Tutorial5Client client = new Tutorial5Client();
3     client.execute();
4 }
5
6 private void execute()
7 {
8     System.out.println("[CLIENT] - Starting Test...");
9     if(isConnected())
10    {
11        System.out.println("[CLIENT] - Server is connected, continuing test...");
12    }
13    else
14    {
15        System.out.println("[CLIENT] - Server is NOT connected, test failed ! ");
16    }
17    System.out.println("[CLIENT] - Test Completed");
18 }
19 }
```

Figure 16, Client code

WARNING – Bad coding detected You should know why 😊

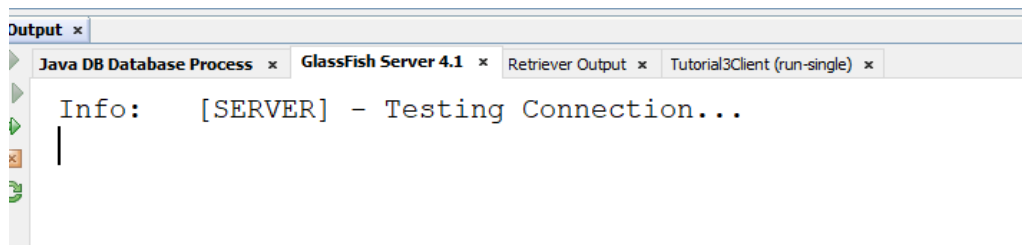
- 16) **ONLY If the client code does not compile because it cannot find the classes in the server package of the client stubs**, you can find the code under Generated Sources (jax-ws). Simply copy and paste the entire java package server into your client code.

- 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 x
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\NetBeansProjec
wsimport-init:
wsimport-client-SimpleWebService:
files are up to date
wsimport-client-generate:
Compiling 1 source file to C:\Users\User\Documents\NetBeansPro
compile-single:
run-single:
[CLIENT] - Starting test...
[CLIENT] - Server is connected, test can proceed...
[CLIENT] - Test Completed !
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 17, Client Log



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

Figure 18, Server Log

- 19) Now, we develop the server further. We want to implement a simple temperature recording system. First we concentrate on how to implement a temperature. A Double type would be a simple choice but it would require to re-write the code if we wanted to add other information (such as time, place and unit of measure). To make our code extendable and maintainable, we create a user-defined type called TemperatureSample which contains a private field Double, getters and setters (Figure 19, TemperatureSample class). You can create automatically getters and setters by right-clicking on the code, insert code, getters and setters. We also add the toString() method for logging reasons.

```
/**
 *
 * @author Gab
 */
public class TemperatureSample {
    Double value = null;

    public Double getValue() {
        return value;
    }

    public void setValue(Double value) {
        this.value = value;
    }

    @Override
    public String toString() {
        return "TemperatureSample{" + "value=" + value + '}';
    }
}
```

Figure 19, TemperatureSample class

JavaBean (coffee anyone ?) = This class is a “JavaBean” class (encapsulation of data with getter and setters) – so you do not use system outs here ! This class does not have any business logic coded. It’s a simple “basket” to collect data. This resides in the server. You will see the client stub would create a “replica” when you re-generate stubs.

- 20) Now we add to the server an addSample Web Method that will allow us to add TemperatureSamples to the server (Figure 20, create WebMethod skeleton of addSample step 1, Figure 21, create WebMethod skeleton of addSample step 2, Figure 22, implement logic of the WebMethod addSample).

Add Operation...

Name:

Return Type:

Parameters Exceptions

Name	Type	Final
sample	server.TemperatureSample	<input type="checkbox"/>

Figure 20, create WebMethod skeleton of addSample step 1

Add Operation...

Name:

Return Type:

Parameters Exceptions

Exception

Figure 21, create WebMethod skeleton of addSample step 2

```
/**
 * Web service operation
 * @param sample
 * @return
 * @throws java.lang.Exception
 */
@WebMethod(operationName = "addSample")
public Boolean addSample(@WebParam(name = "sample") server.TemperatureSample sample) throws Exception {

    if(sample == null)
        throw new Exception();

    System.out.println("[SERVER] - addSample(" + sample + ") to " + samples);
    samples.add(sample);
    System.out.println("[SERVER] - Now samples are " + samples);
    return true;
}
```

Figure 22, implement logic of the WebMethod addSample

- 21) To store the Samples on the server, we declare an ArrayList of Temperature Samples (Figure 23, Declaring an ArrayList on the Server) which we use in the addSample Web Method.

```
*/
@WebService(serviceName = "Tutorial5WebService")
public class Tutorial5WebService
{
    ArrayList<TemperatureSample> samples = new ArrayList();
}
```

Figure 23, Declaring an ArrayList on the Server

- 22) Add another Web Method: getNumberOfSamples which returns the size of the samples ArrayList, we will use it to test the Server. (Figure 24, create and implement the getNumberOfSamples Web Method.)

```
/**
 * Web service operation
 * @return
 */
@WebMethod(operationName = "getNumberOfSamples")
public Integer getNumberOfSamples() {
    System.out.println("[SERVER] - getNumberOfSamples()");
    return samples.size();
}
```

Figure 24, create and implement the getNumberOfSamples Web Method.

- 23) Finally, create and implement the getMaximum Web Method which returns the maximum of all samples. (Figure 25, create and implement the getMaximum Web Method.)

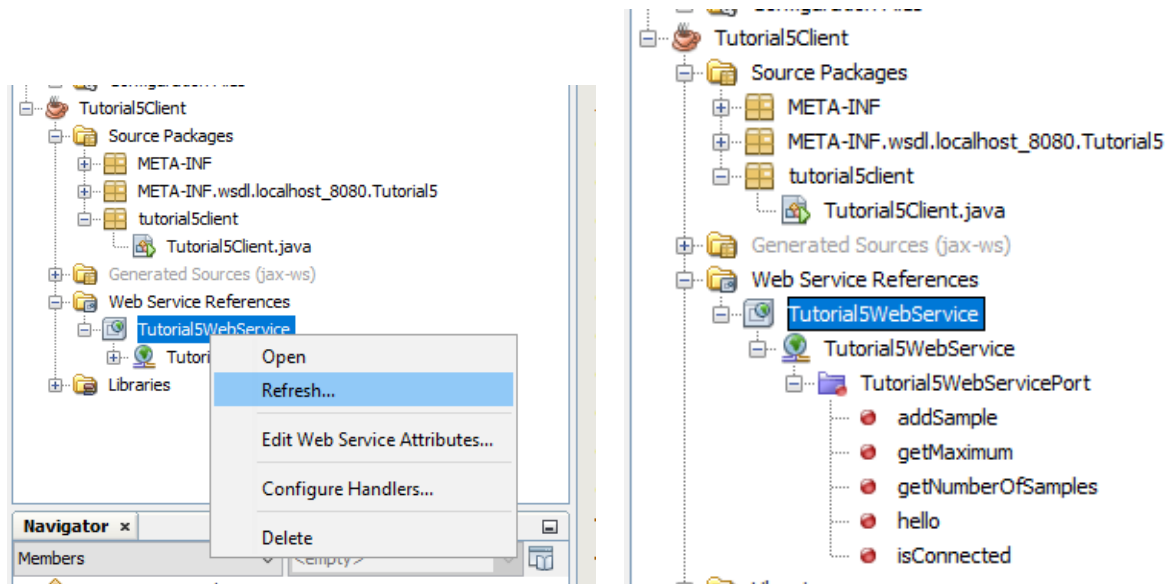
```
*/
@WebMethod(operationName = "getMaximum")
public TemperatureSample getMaximum() throws Exception {
    if(samples.isEmpty())
        throw new Exception();

    TemperatureSample max = new TemperatureSample();
    max = samples.get(0);
    for(int i = 0; i < samples.size(); i++)
        if(samples.get(i).getValue() > max.getValue())
            max = samples.get(i);

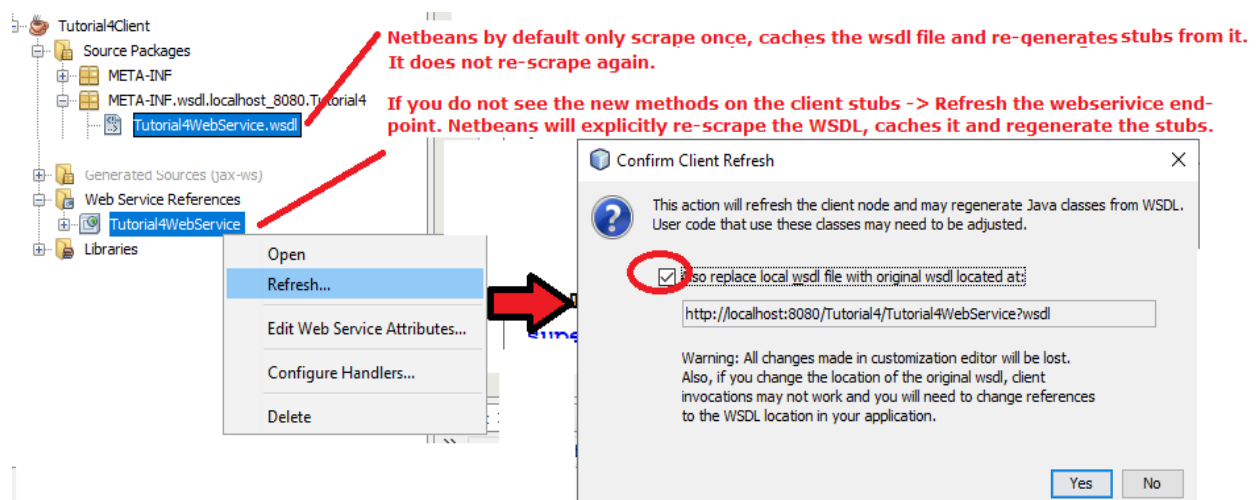
    return max;
}
```

Figure 25, create and implement the getMaximum Web Method.

- 24) Now we want to test the new Web Server functionalities from the Client. First we must deploy the server, then refresh the Client Stubs. You can observe that the Web Service Reference has been updated.



Netbeans caches the WSDL = if you do not see your “new” methods – you need to explicitly refresh the client stubs to tell Netbeans to re-scrape the updated wsdl.



- 25) Copy the client stubs into your client code and invoke them. (Figure 26, Test the new Web Methods from the client.) **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 (jaws). Simply delete the server package in your source code and re-paste the entire java package server into from generated sources into your client code.

```
if (isConnected()) {
    System.out.println("[CLIENT] - Server is connected, continuing test...");
    TemperatureSample s = new TemperatureSample();
    s.setValue(0.1);
    try {
        addSample(s);
        System.out.println("[CLIENT] - Server has " + getNumberOfSamples() + " sampl
        System.out.println("[CLIENT] - Maximum sample on the Server is " + getMaximu

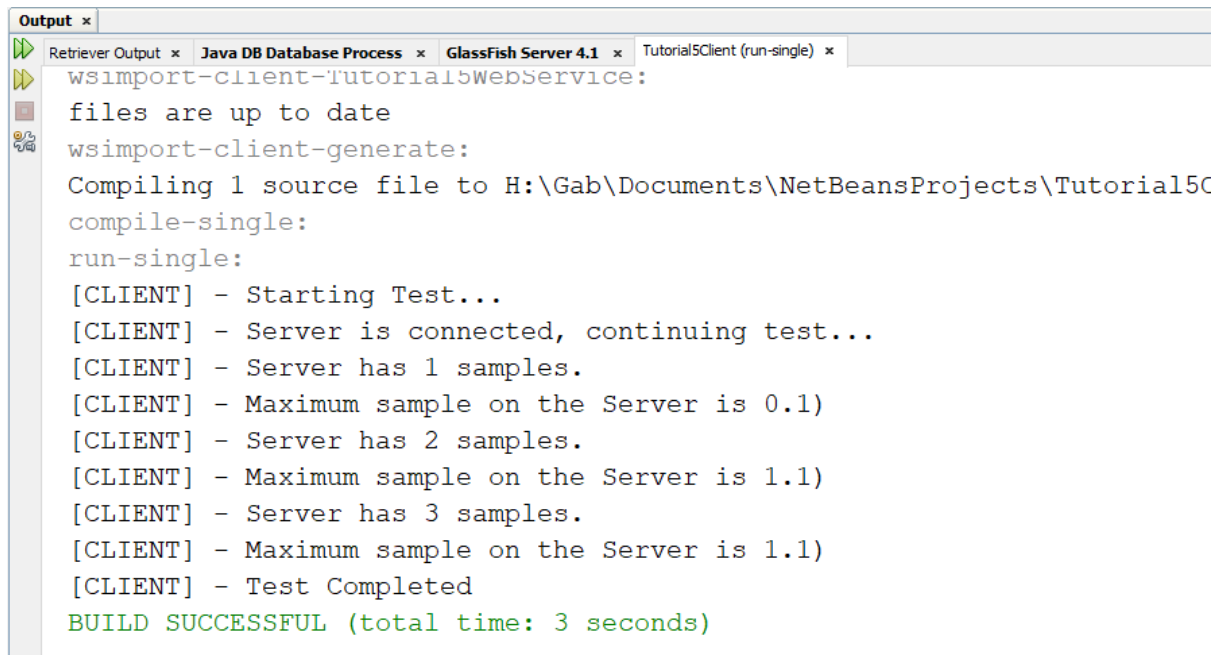
    } catch (Exception_Exception ex) {
        Logger.getLogger(Tutorial5Client.class.getName()).log(Level.SEVERE, null, ex
    }
    s.setValue(1.1);
    try {
        addSample(s);
        System.out.println("[CLIENT] - Server has " + getNumberOfSamples() + " sampl
        System.out.println("[CLIENT] - Maximum sample on the Server is " + getMaximu
    } catch (Exception_Exception ex) {
        Logger.getLogger(Tutorial5Client.class.getName()).log(Level.SEVERE, null, ex
    }

    s.setValue(-1.1);
    try {
        addSample(s);
        System.out.println("[CLIENT] - Server has " + getNumberOfSamples() + " sampl
        System.out.println("[CLIENT] - Maximum sample on the Server is " + getMaximu
    } catch (Exception_Exception ex) {}
        Logger.getLogger(Tutorial5Client.class.getName()).log(Level.SEVERE, null, ex
    }
}
```

Figure 26, Test the new Web Methods from the client.

YOU DO THIS COPYING OF STUBS BUSINESS LAST = After copying do not rebuild. Simply right click the file and select run. If you rebuild, it will throw a duplicate class error. If you need to rebuild, delete the copied stubs (except your main file!).

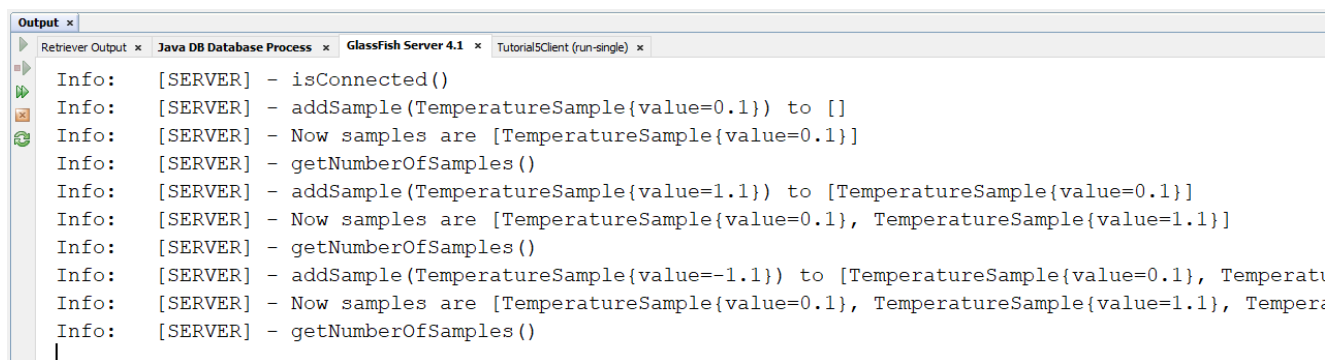
- 26) Test the client and check the logs on both sides: client and server.



The screenshot shows the NetBeans IDE's Output window with the following tabs: Retriever Output, Java DB Database Process, GlassFish Server 4.1, and Tutorial5Client (run-single). The selected tab, Tutorial5Client (run-single), displays the following log output:

```
wsimport-client-Tutorial5WebService:
files are up to date
wsimport-client-generate:
Compiling 1 source file to H:\Gab\Documents\NetBeansProjects\Tutorial5C
compile-single:
run-single:
[CLIENT] - Starting Test...
[CLIENT] - Server is connected, continuing test...
[CLIENT] - Server has 1 samples.
[CLIENT] - Maximum sample on the Server is 0.1)
[CLIENT] - Server has 2 samples.
[CLIENT] - Maximum sample on the Server is 1.1)
[CLIENT] - Server has 3 samples.
[CLIENT] - Maximum sample on the Server is 1.1)
[CLIENT] - Test Completed
BUILD SUCCESSFUL (total time: 3 seconds)
```

Figure 27, Client Log



The screenshot shows the NetBeans IDE's Output window with the same tabs as Figure 27. The selected tab, Tutorial5Client (run-single), displays the following log output:

```
Info: [SERVER] - isConnected()
Info: [SERVER] - addSample(TemperatureSample{value=0.1}) to []
Info: [SERVER] - Now samples are [TemperatureSample{value=0.1}]
Info: [SERVER] - getNumberOfSamples()
Info: [SERVER] - addSample(TemperatureSample{value=1.1}) to [TemperatureSample{value=0.1}]
Info: [SERVER] - Now samples are [TemperatureSample{value=0.1}, TemperatureSample{value=1.1}]
Info: [SERVER] - getNumberOfSamples()
Info: [SERVER] - addSample(TemperatureSample{value=-1.1}) to [TemperatureSample{value=0.1}, Temperat
Info: [SERVER] - Now samples are [TemperatureSample{value=0.1}, TemperatureSample{value=1.1}, Temper
Info: [SERVER] - getNumberOfSamples()
```

Figure 28, Server Log

- 27) Note that if you re-execute the client, the number of sample of the server grows (the server is now stateful). To reset the server, redeploy it.
- 28) Finally, export both the projects (client and server) as zip file on Netbeans.

TASKS to BE PERFORMED Independently by the student (from Task 28 to Task 31) (Formative Assessment)

- 29) Add a Web Method that returns the minimum temperature and test it from the client.
- 30) Add a Web Method that returns the average temperature and test it from the client.
- 31) A Temperature Sample is not meaningful without the time and place it has been taken from. Modify the TemperatureSample class (now you can understand why it is useful to have our model in a class) so it contains the time when the sample was taken and the place (as a string).