University of Westminster
Faculty of Science and Technology

Module code and title: 5COSC004W-Client Service Architecture Tutorial Manual

Tutorial title	**Developing a real Client/Server system based on SOAP technology**
Tutorial type	**Guided and indepenent and non-marked**
Week 05	**20/02/2020**

Contents

Learning Goals

This tutorial focuses on the fundamental learning goal to create a REAL Client/Server system based on the SOAP Web Service Technology.
While the previous tutorials (1 to 4) were based on a Dummy Client/Server Architecture were both the server and the client were just two classes in a single project. Now we build a REAL web service that will execure the client and the server as SEPARATE programs which communicate through a communication technology called Web Services (Figure 1, Tutorial 1 to 4 and Tutorial 5 Architectures). This actually means that the execution environments of the Client and the Server will be separate and we MUST use a proper technology to transfer information from one to the other. As an example, you will not be able in the Client to invoke directly a method declared on the Server.

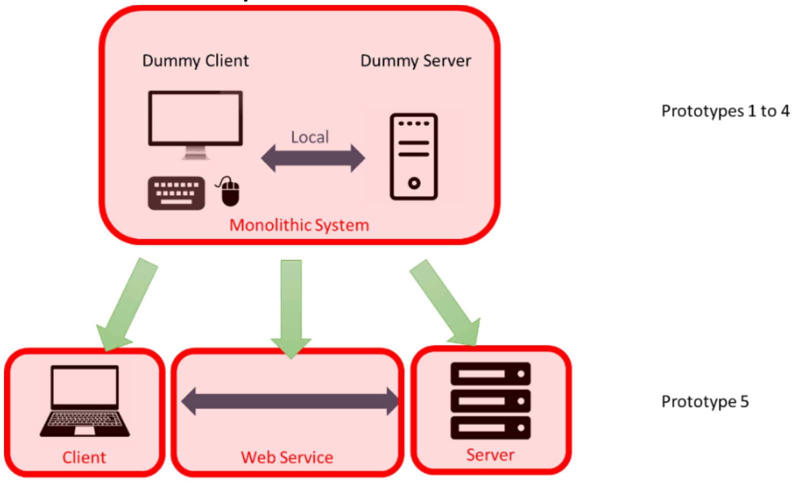


Figure 1, Tutorial 1 to 4 and Tutorial 5 Architectures

Now, in order to do that, we have to invoke an internet protocol (TCP, UDP or IP) to transfer data. To do that directly (e.g. Sockets) it takes a significant effort and the Scientific Community has developed automated code generation mechanisms that will create all this code for you. We will use Java-WS which uses the SOAP and WSDL languages to stream information between the Client and the Server.
They way this works is that (, first we will create an Interface to the Server which makes it available to the Internet and then we will create a series of Client Stubs that will allow the Client to invoke the remote server as if it was a local class.

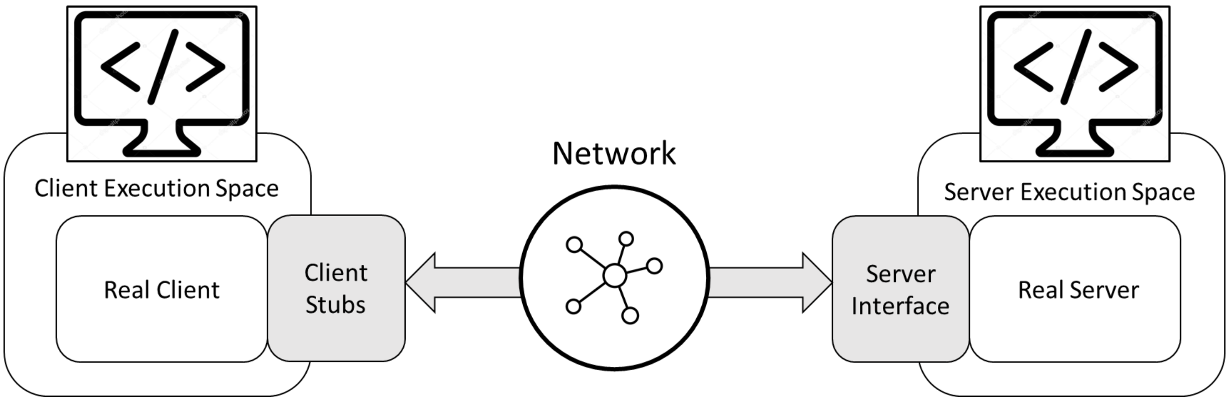
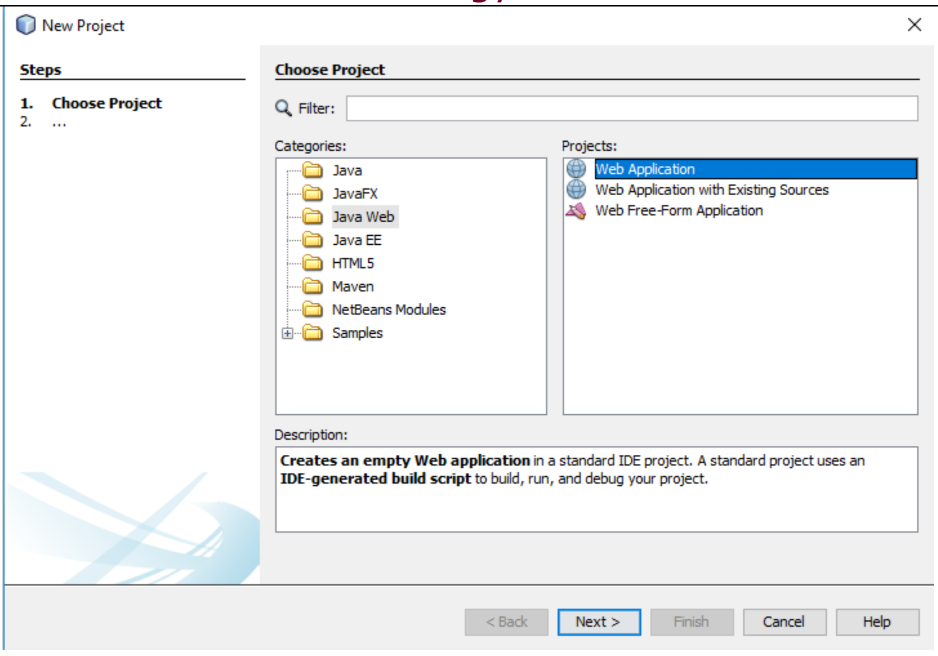


Figure 2, Client Stubs and Server Interfaces

As usual, the tutorial it is divided into two separate sections, the student will perform the first task (1- 23) following the instructions of the tutor, and then, will complete the other tasks independently.

TASKS to be Performed under the instruction of the Tutor (from Task 1 to Task 23)

1) Start Netbeans in your system. If Netbeans is not present in your system, use AppsAnywhere to launch it: (https://www.westminster.ac.uk/sites/default/public-files/general-documents/Using%20AppsAnywhere.pdf)
2) **Instead of creating a new java project, this time we create a new Web Application Project** (Figure 3Figure 3, Create new Web Application Project (Step 1)


New Project

Steps
1. Choose Project
2. ...

Choose Project

Filter:

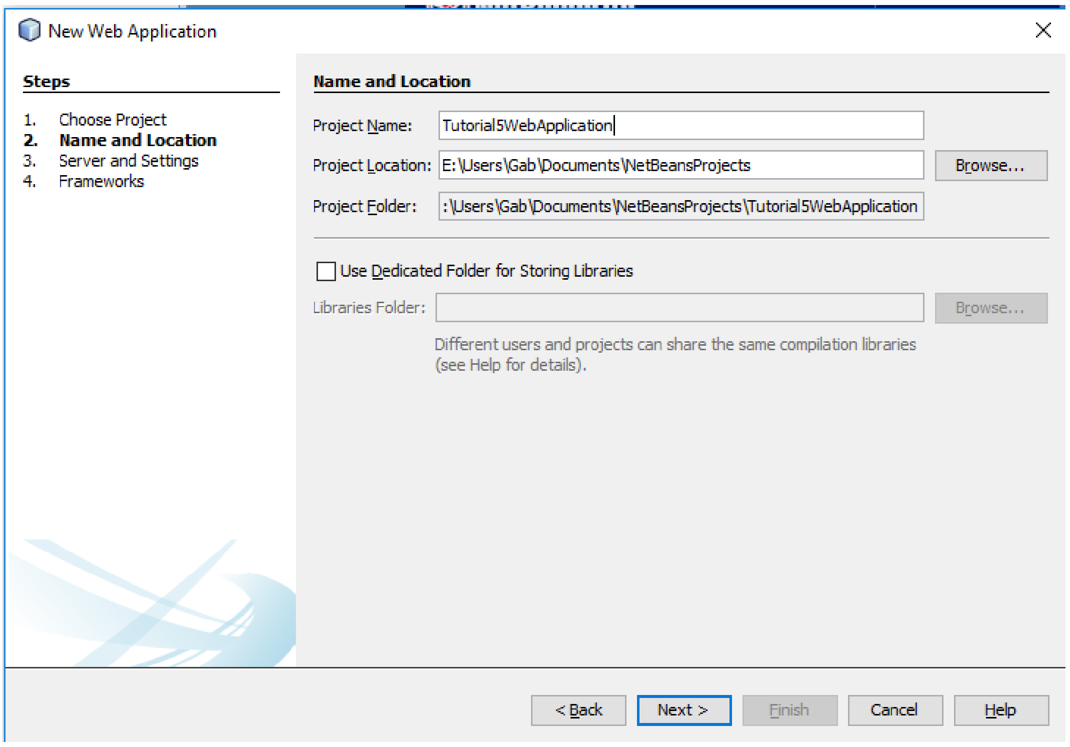
Categories: Java, JavaFX, Java Web, Java EE, HTML5, Maven, NetBeans Modules, Samples

Projects: Web Application, Web Application with Existing Sources, Web Free-Form Application

Description:
Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

Figure 3, Create new Web Application Project (Step 1)

3) Let's call our first Web Application Tutorial5WebApplication (Figure 4)


New Web Application

Steps
1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: Tutorial5WebApplication

Project Location: E:\Users\Gab\Documents\NetBeansProjects

Project Folder: :\Users\Gab\Documents\NetBeansProjects\Tutorial5WebApplication

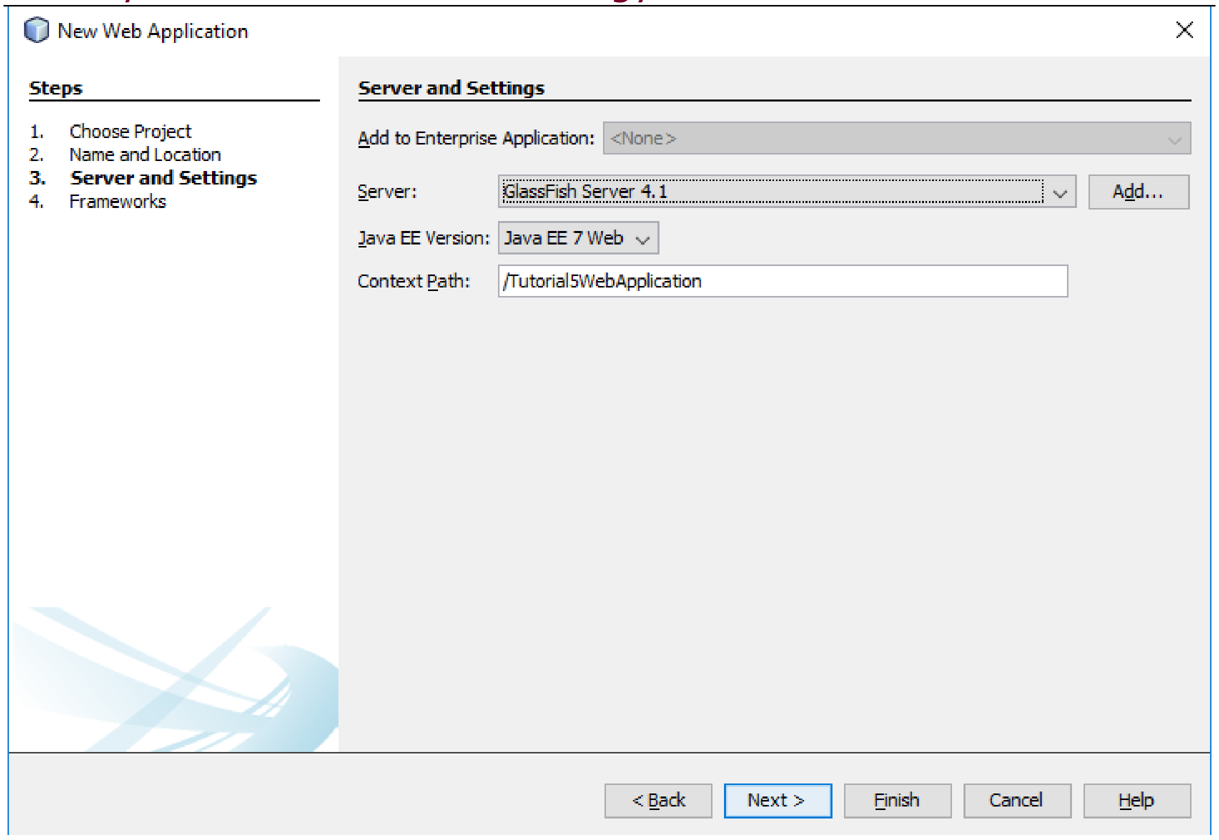
Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

Figure 4,Create new Web Application Project (Step 2)

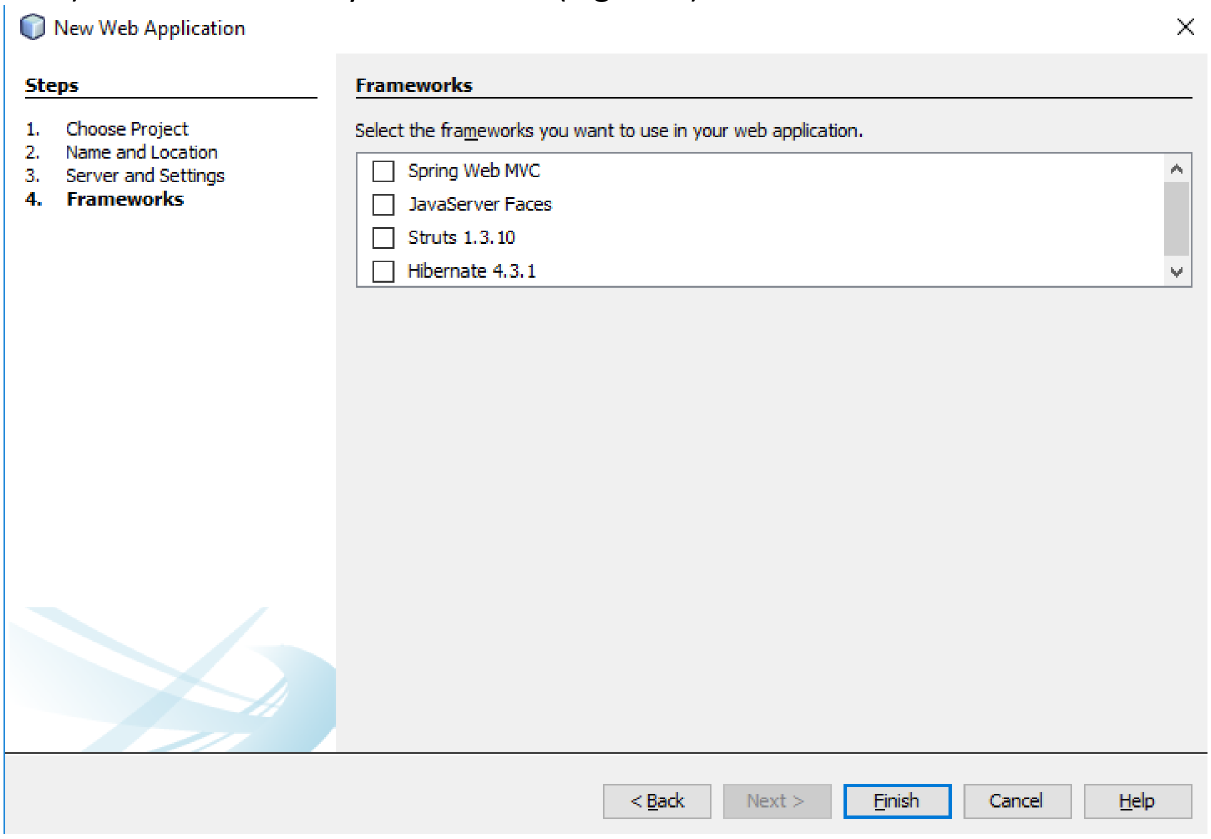
4) Now, this time we are building a REAL server, so we neet to select the Server Engine we would like to use, select GlassFish (Figure 5)



Add to Enterprise Application: <None>
Server: GlassFish Server 4.1
Java EE Version: Java EE 7 Web
Context Path: /Tutorial5WebApplication

Figure 5,Create new Web Application Project (Step 3)

5) Do not select any framework. (Figure 6)

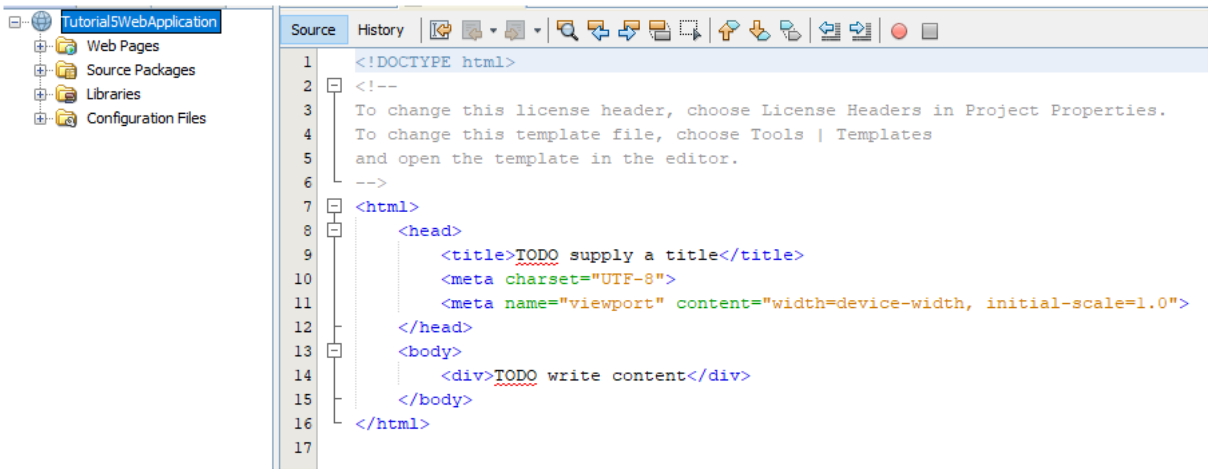


Select the frameworks you want to use in your web application.

- [ ] Spring Web MVC
- [ ] JavaServer Faces
- [ ] Struts 1.3.10
- [ ] Hibernate 4.3.1

Figure 6, Create new Web Application Project (Step 4)

6) We did not specify anything about our Web Application, so Netbeans has creates a simple Web Page (Figure 7).



```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div>TODO write content</div>
    </body>
</html>
```

Figure 7, Create new Web Application Project (Step 5)

7) You can even test this web page by running the Web Application, right click on the project and select run (Figure 8) and Figure 9. Our Web Application at the moment is just a simple Web Page.

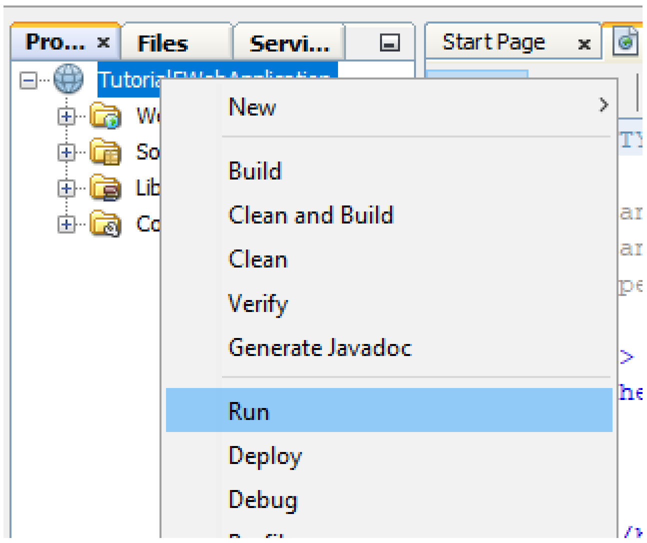
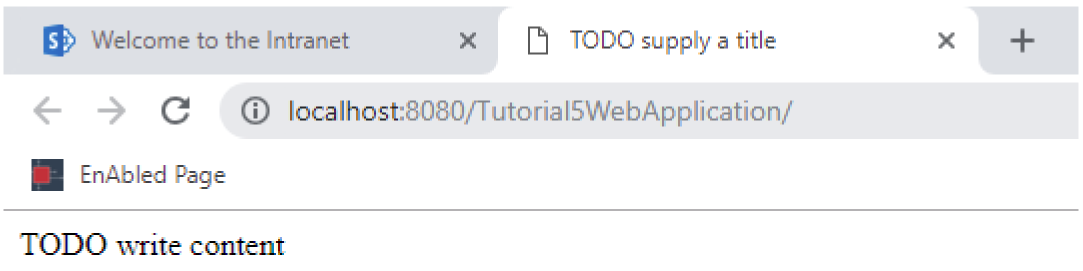


Figure 8, Running the new Web Application Project (Step 1)



TODO write content

Figure 9, Running the new Web Application Project (Step 2)

8) Create a new Java package in the Web Application where we will put our server, called it server (Figure 10) and (Figure 11Figure 11,Creating a server Java Package (Step 2))

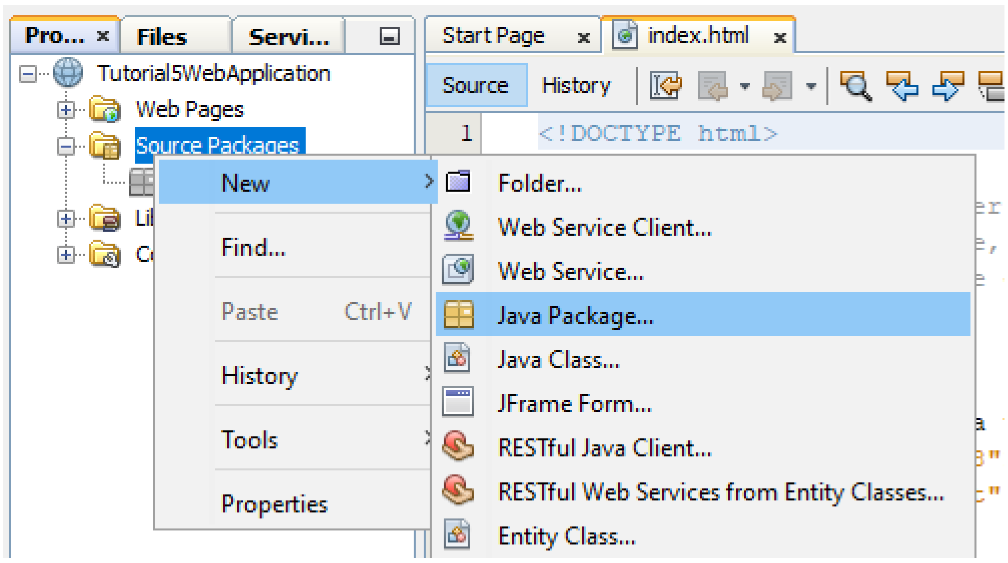
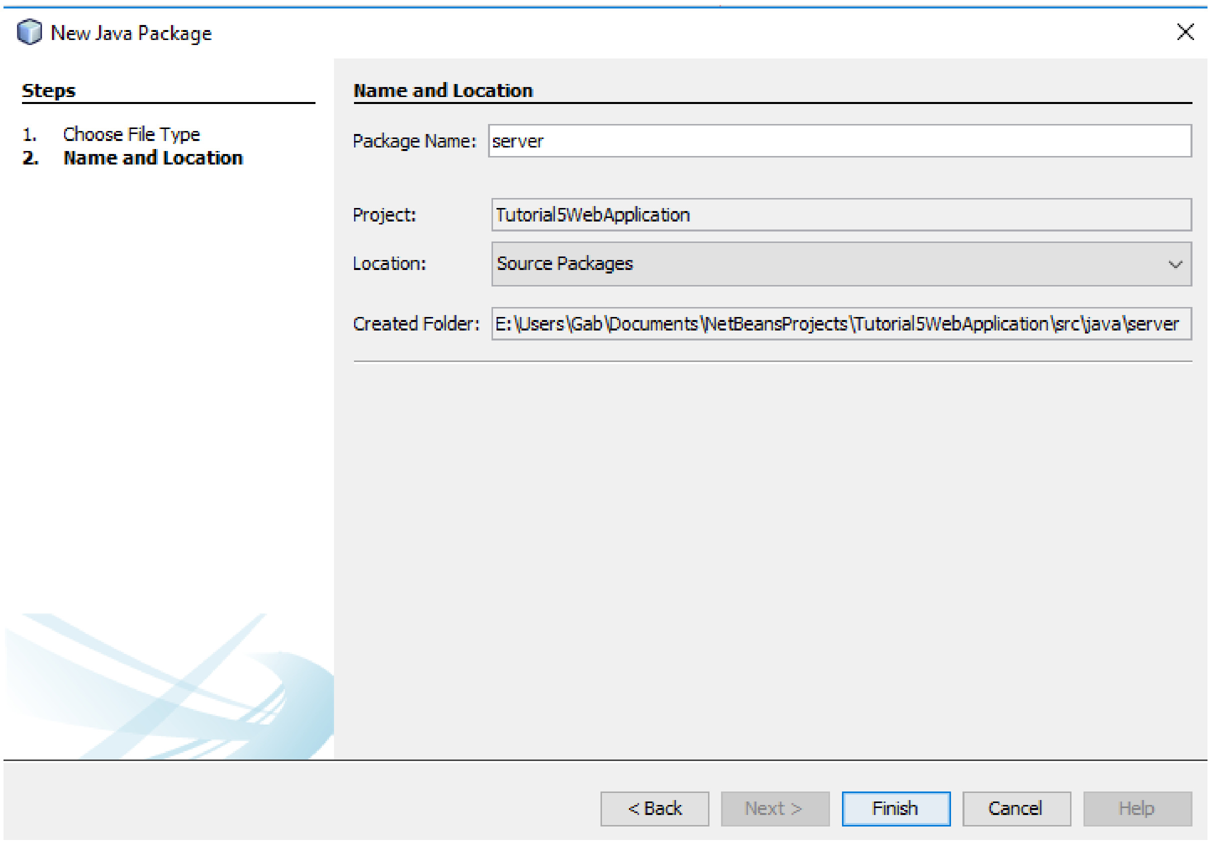


Figure 10, Creating a server Java Package (Step 1)


New Java Package

Steps

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

Name and Location

Package Name: server

Project: Tutorial5WebApplication

Location: Source Packages

Created Folder: E:\Users\Gab\Documents\NetBeansProjects\Tutorial5WebApplication\src\java\server

< Back | Next > | Finish | Cancel | Help

Figure 11,Creating a server Java Package (Step 2)

9) Now, we can create a proper Web Service, we will call it SimpleWebService (Figure 12) and (Figure 13)

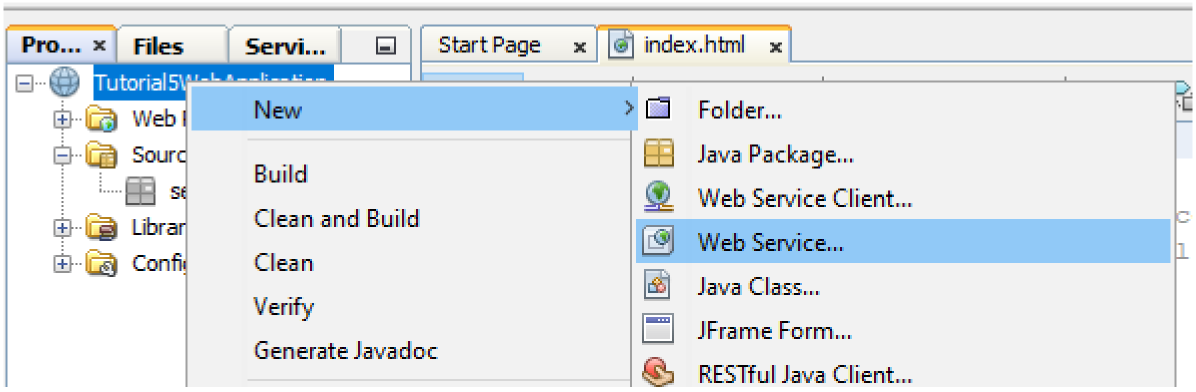
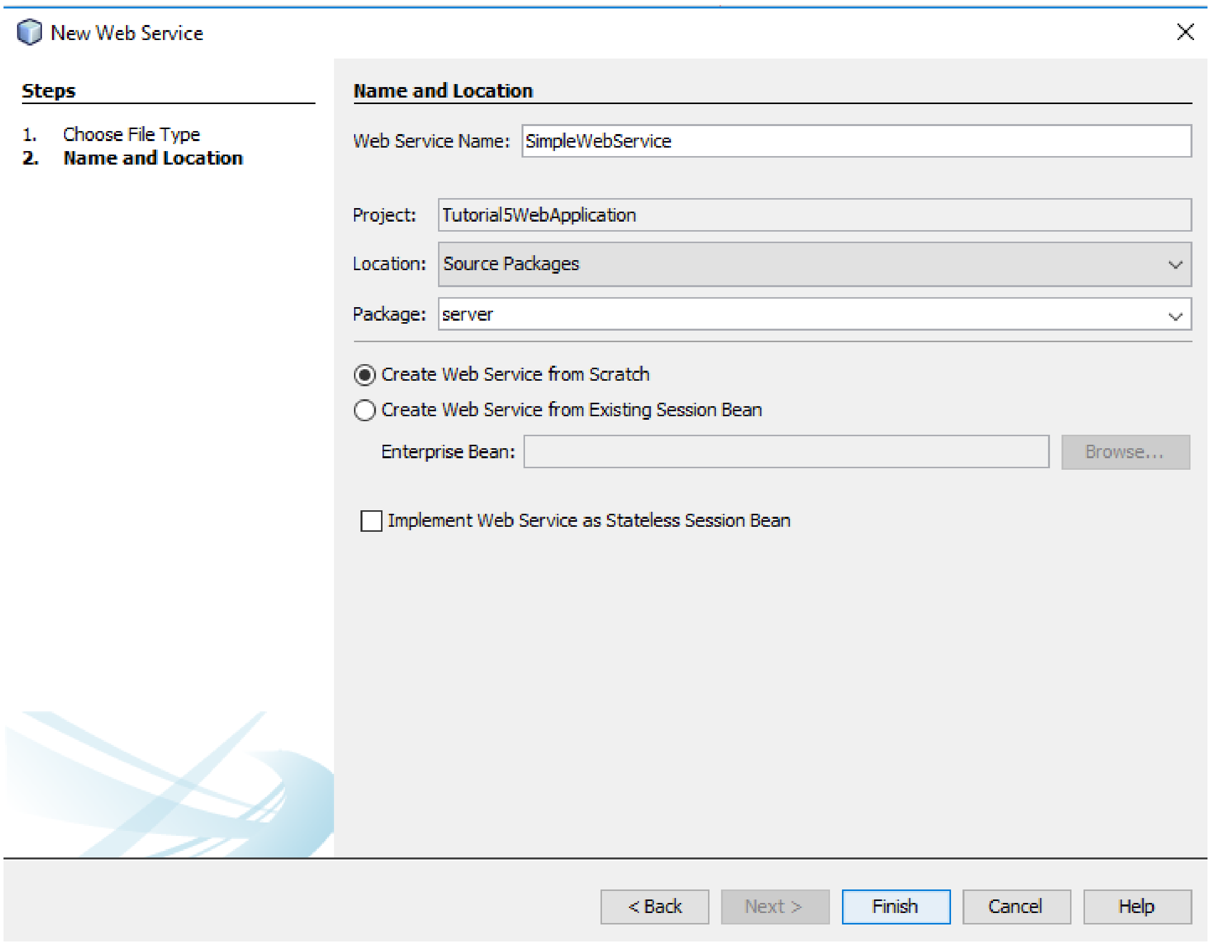


Figure 12, Create the Web Service (step 1)



Web Service Name: SimpleWebService

Project: Tutorial5WebApplication

Location: Source Packages

Package: server

(●) Create Web Service from Scratch

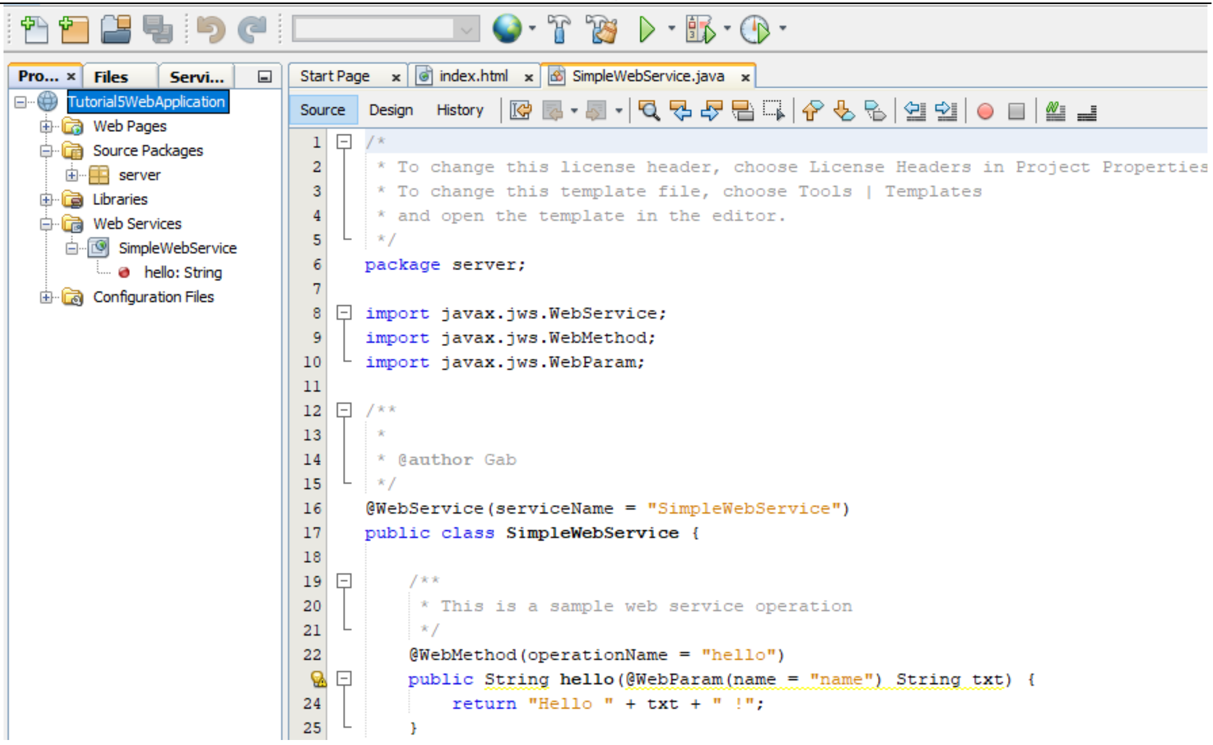
( ) Create Web Service from Existing Session Bean

Enterprise Bean:

☐ Implement Web Service as Stateless Session Bean

Figure 13,Create the Web Service (step 2)

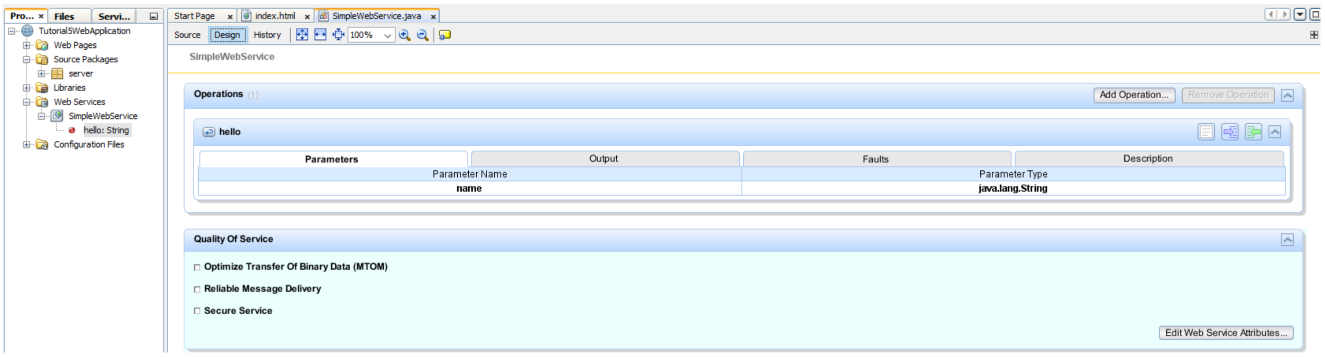
10) You can observe that NetBeans creates a standar method called Hello which echoes the message sent. (Figure 14)



```java
/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package server;

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

/**
 *
 * @author Gab
 */
@WebService(serviceName = "SimpleWebService")
public class SimpleWebService {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }
```

Figure 14, Standard Web Service Created by Netbeans.

11) 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 (Figure 15)

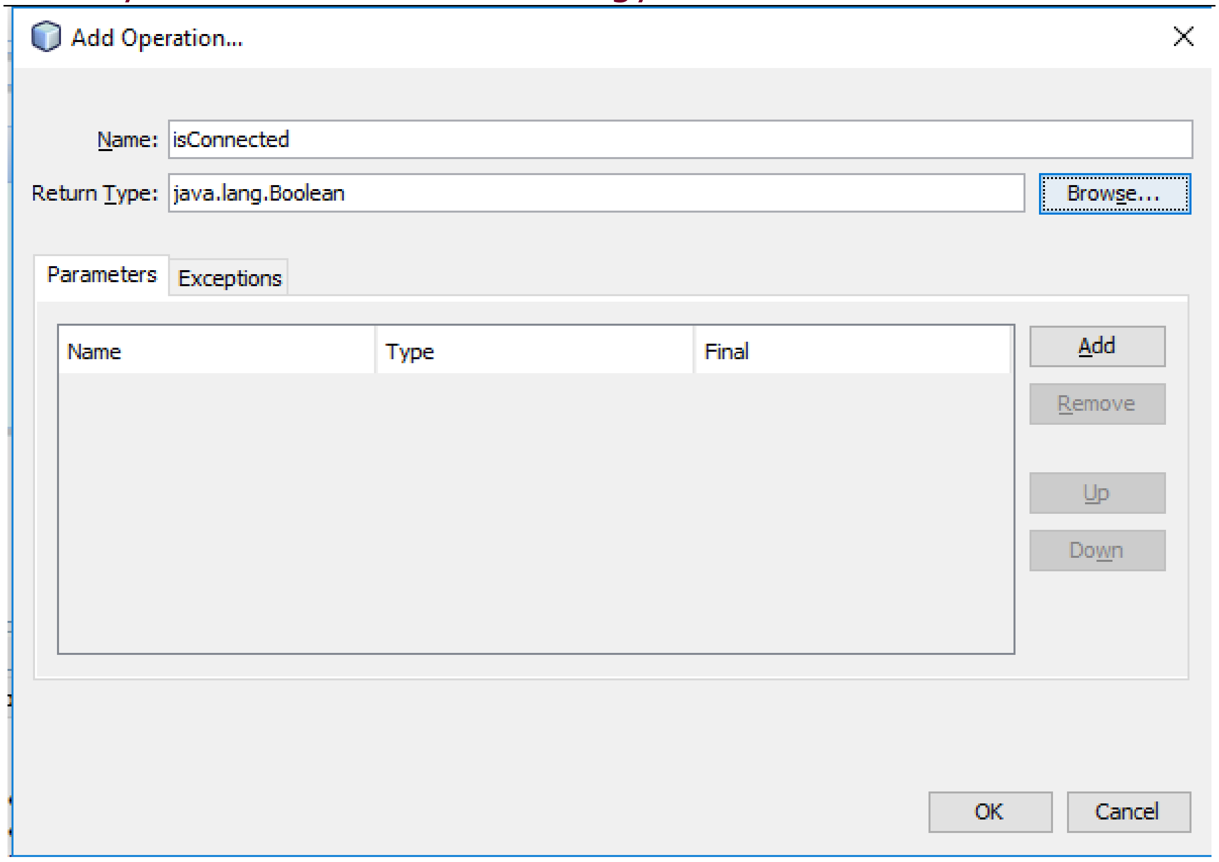


Parameter Name	Parameter Type
name	java.lang.String

☐ Optimize Transfer Of Binary Data (MTOM)
☐ Reliable Message Delivery
☐ Secure Service

Figure 15, Design view of the Web Service

12) We use the AddOperation Button to add our usual isConnected methods that returns true if the server is connected (Figure 16) and (Figure 17).

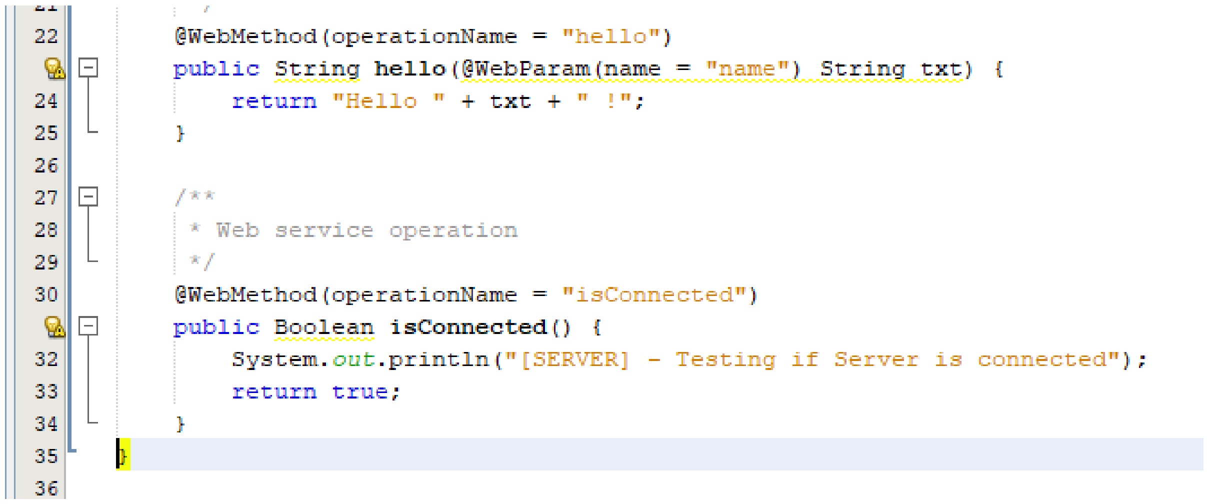


Name: isConnected

Return Type: java.lang.Boolean [Browse...]

Name	Type	Final

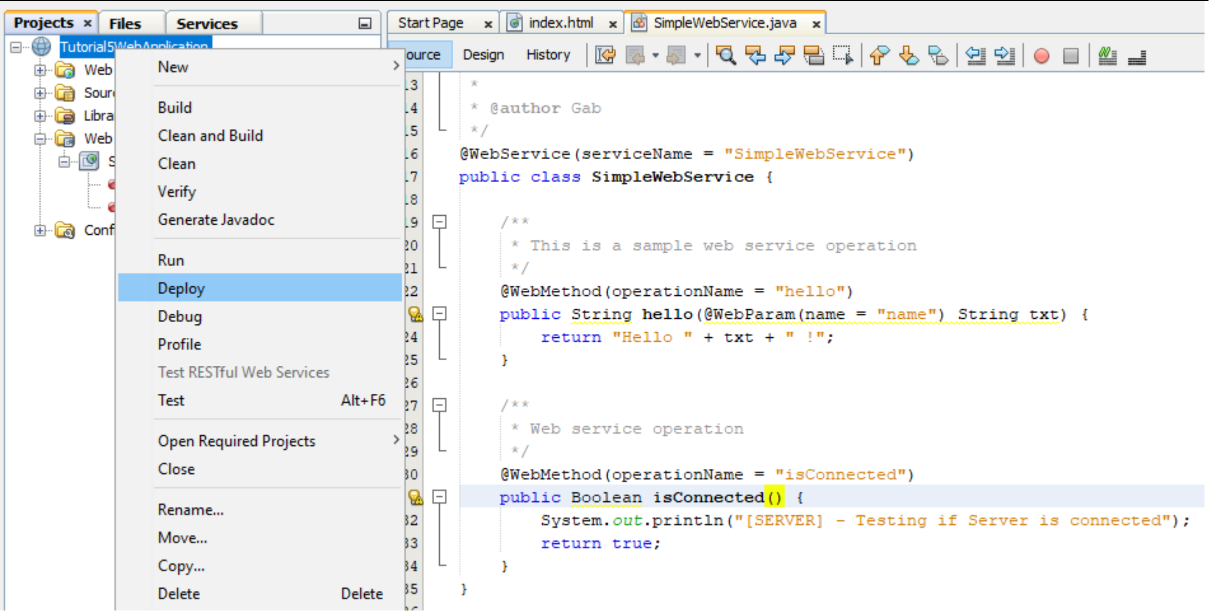
Figure 16, Adding the isConnected method to the Server (step 1)



```
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "isConnected")
    public Boolean isConnected() {
        System.out.println("[SERVER] - Testing if Server is connected");
        return true;
    }
}
```

Figure 17, Adding the isConnected method to the Server (step 2)

13) Now we deploy the server on the glassfish server engine (Figure 18)



```
 *
 * @author Gab
 */
@WebService(serviceName = "SimpleWebService")
public class SimpleWebService {

    /**
     * This is a sample web service operation
     */
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }

    /**
     * Web service operation
     */
    @WebMethod(operationName = "isConnected")
    public Boolean isConnected() {
        System.out.println("[SERVER] - Testing if Server is connected");
        return true;
    }
}
```

Figure 18, Deploying the Server

14) NetBeans offers us a useful testing tool without even to have to write a client (NetBeans will write a simple client in a browser)(Figure 19). Test the isConnected method, it returns true, that is correct (Figure 20) and Figure 21! You can also check the server log to see if it is correct (Figure 22)

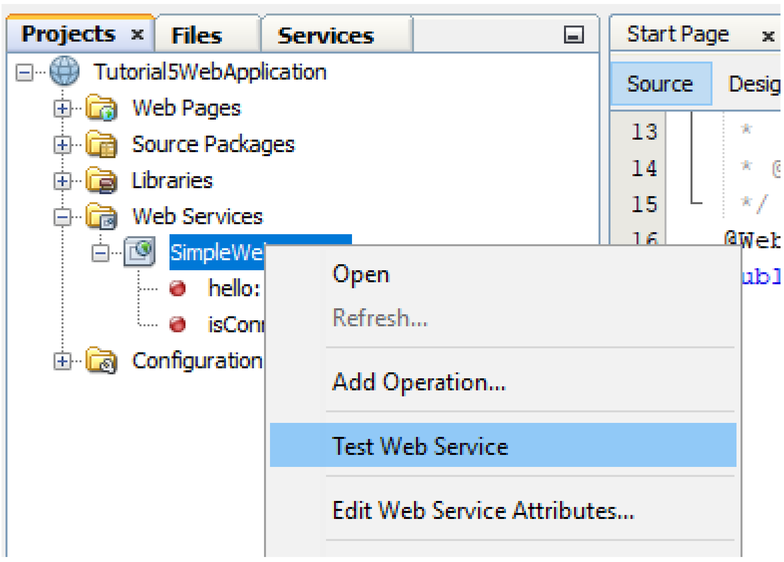
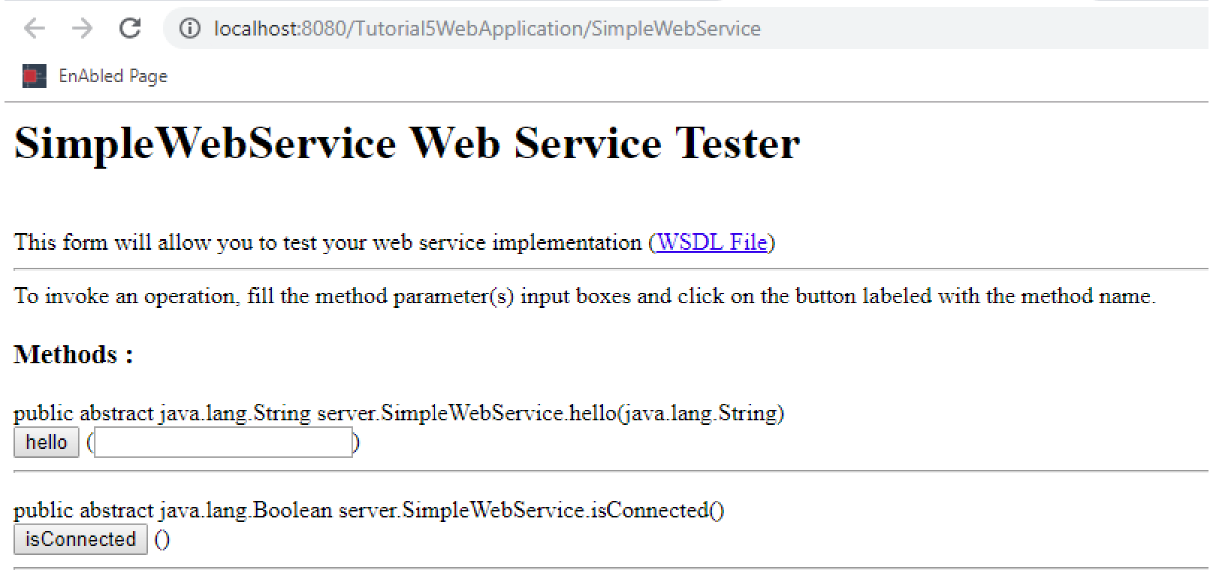


Figure 19, Testing the Web Service (Step 1)



SimpleWebService Web Service Tester

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

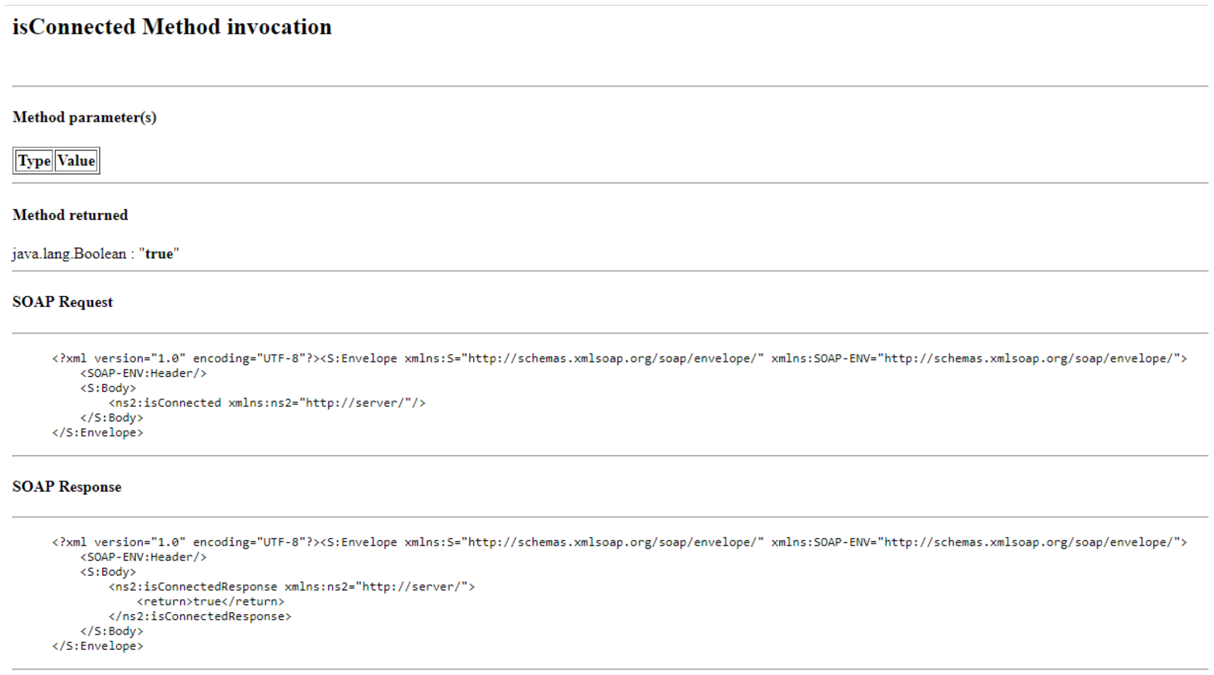
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

public abstract java.lang.String server.SimpleWebService.hello(java.lang.String)

public abstract java.lang.Boolean server.SimpleWebService.isConnected()

Figure 20,Testing the Web Service (Step 2)



isConnected Method invocation

Method parameter(s)

Type	Value

Method returned

java.lang.Boolean : "**true**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:isConnected xmlns:ns2="http://server/"/>
    </S:Body>
</S:Envelope>
```

SOAP Response

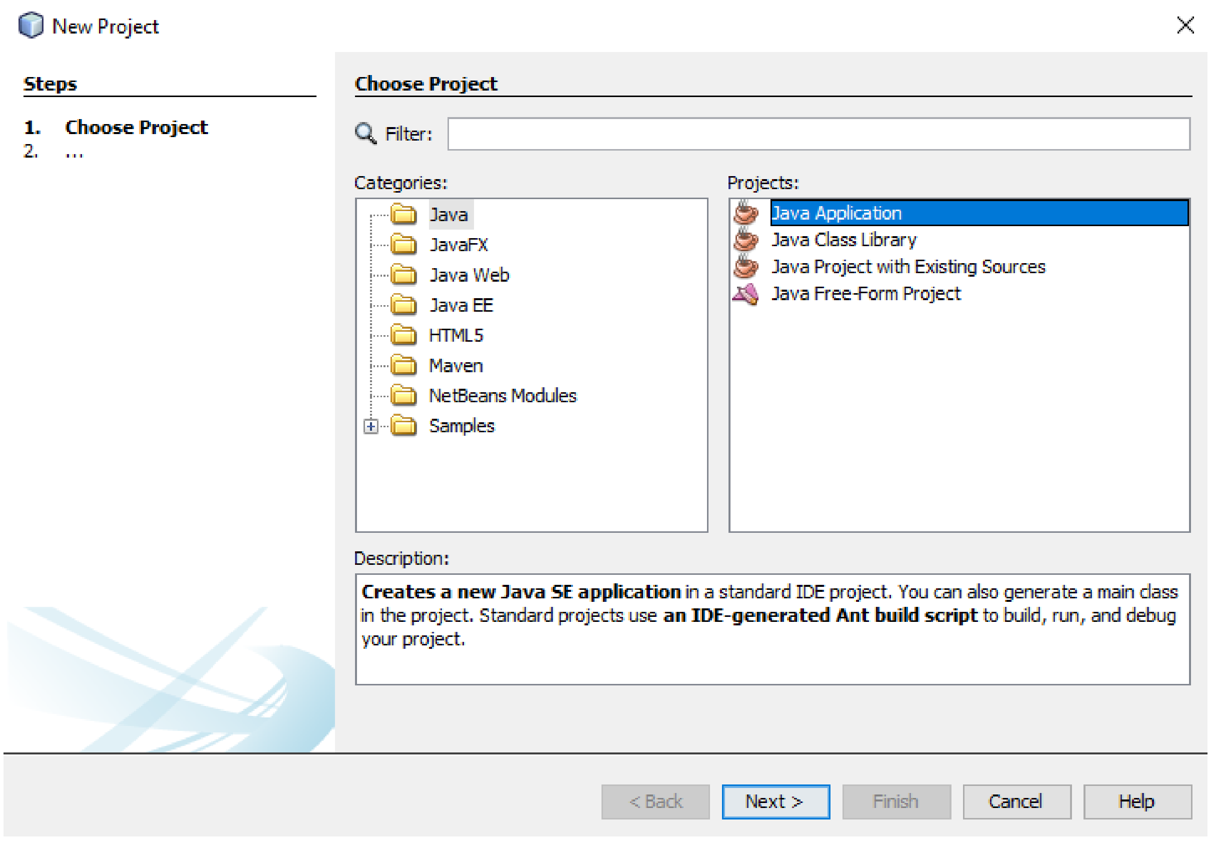
```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body>
        <ns2:isConnectedResponse xmlns:ns2="http://server/">
            <return>true</return>
        </ns2:isConnectedResponse>
    </S:Body>
</S:Envelope>
```

Figure 21, Testing the Web Service (Step 3)

```
Info:   Loading application [Tutorial5WebApplication] at [/Tutorial5WebApplication]
Info:   Tutorial5WebApplication was successfully deployed in 1,167 milliseconds.
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   Webservice Endpoint deployed SimpleWebService
 listening at address at http://DESKTOP-V5I2H6T:8080/Tutorial5WebApplication/SimpleWebService.
Info:   Loading application [Tutorial5WebApplication] at [/Tutorial5WebApplication]
Info:   Tutorial5WebApplication was successfully deployed in 567 milliseconds.
Info:   Invoking wsimport with http://localhost:8080/Tutorial5WebApplication/SimpleWebService?WSDL
Info:   parsing WSDL...
Info:   Generating code...
Info:   Compiling code...
Info:   wsimport successful
Info:   Invoking wsimport with http://localhost:8080/Tutorial5WebApplication/SimpleWebService?WSDL
Info:   parsing WSDL...
Info:   Generating code...
Info:   Compiling code...
Info:   wsimport successful
Info:   [SERVER] - Testing if Server is connected
```

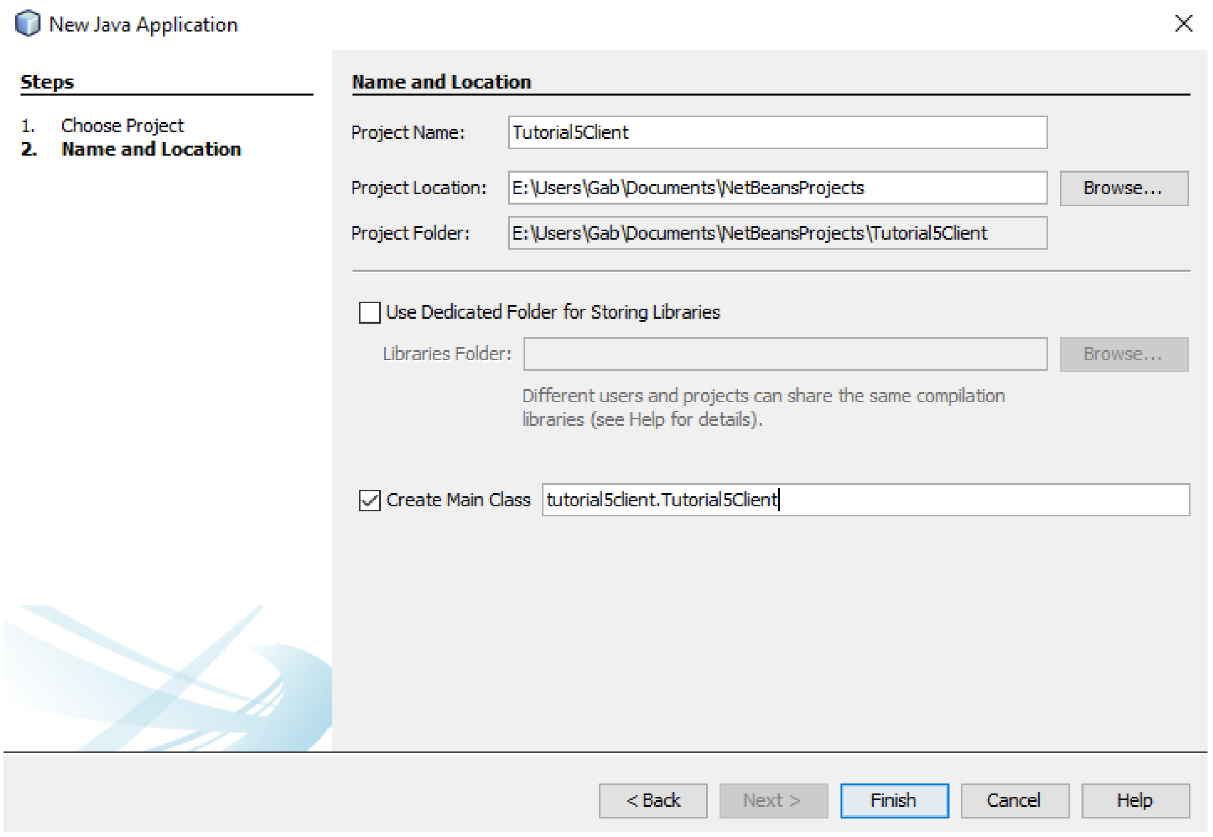
Figure 22, Test Server Log

15) Now we create a client, this is going to be a separate Java Application Project (Figure 23 and Figure 24).



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

Figure 23, Creating the Client (Step 1)



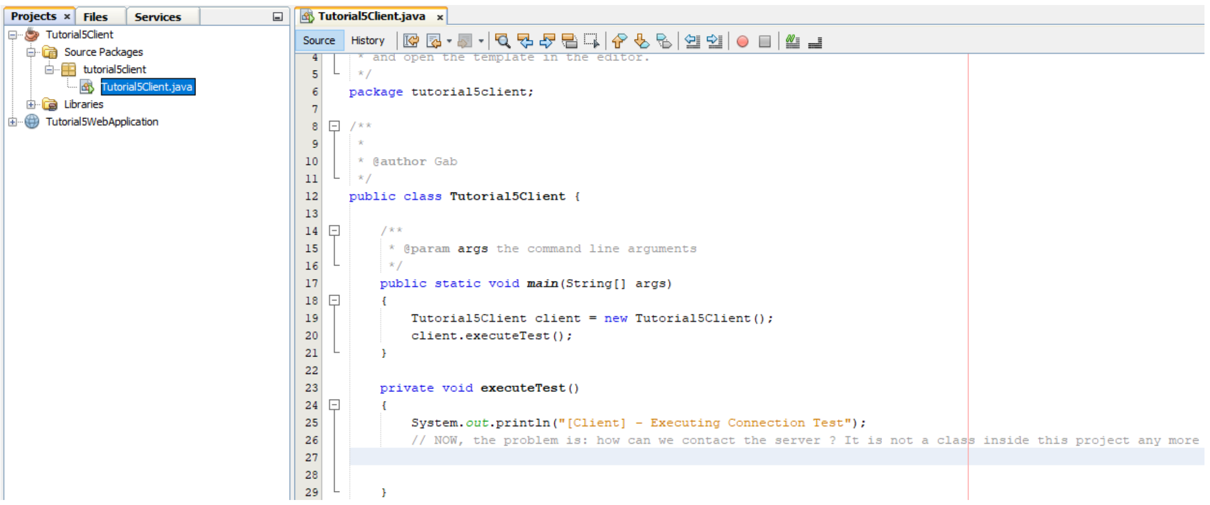
Project Name: Tutorial5Client
Project Location: E:\Users\Gab\Documents\NetBeansProjects
Project Folder: E:\Users\Gab\Documents\NetBeansProjects\Tutorial5Client

☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class tutorial5client.Tutorial5Client

Figure 24,Creating the Client (Step 2)

16) Now we have a problem ! How can we access the server to test if it is connected (Figure 25) ? In the past tutorial it was not a real server so it was just a class in the same project, but now the server is in a separate project. We have to build a communication system between the client and the server !



```
 * and open the template in the editor.
 */
package tutorial5client;

/**
 *
 * @author Gab
 */
public class Tutorial5Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        Tutorial5Client client = new Tutorial5Client();
        client.executeTest();
    }

    private void executeTest()
    {
        System.out.println("[Client] - Executing Connection Test");
        // NOW, the problem is: how can we contact the server ? It is not a class inside this project any more


    }
```

Figure 25, How can we connected to the Server ?

17) 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 (Figure 26, Figure 27, Figure 28 and Figure 29)

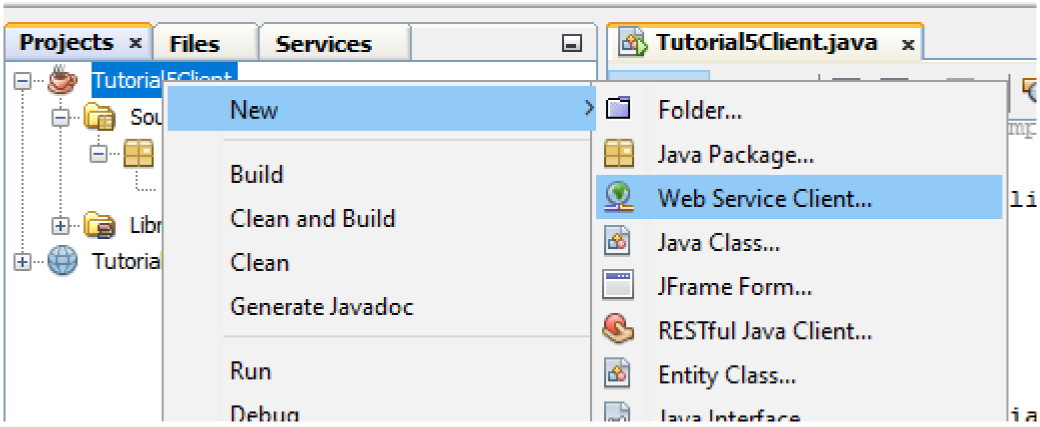
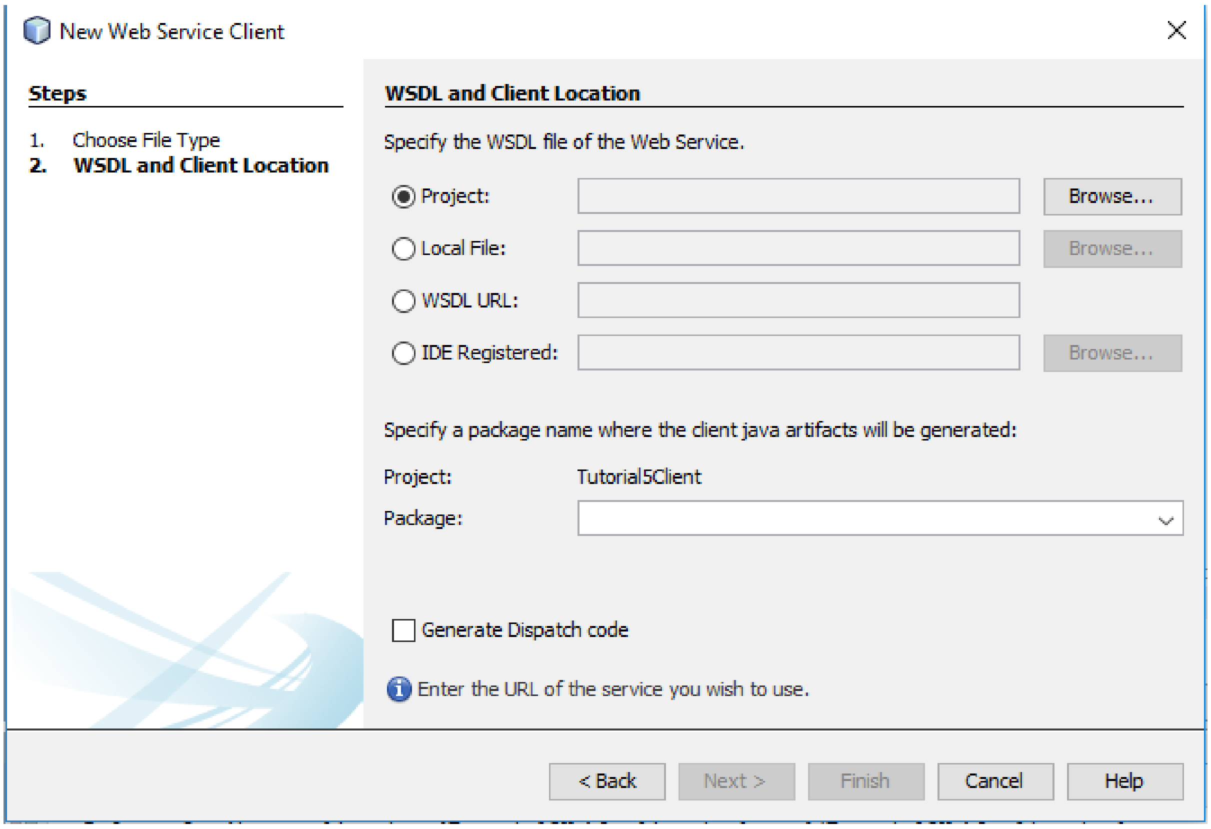


Figure 26, Creating the Web Service Client (Step 1)


New Web Service Client

Steps

1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

- (●) Project: [ ] Browse...
- ( ) Local File: [ ] Browse...
- ( ) WSDL URL: [ ]
- ( ) IDE Registered: [ ] Browse...

Specify a package name where the client java artifacts will be generated:

Project: Tutorial5Client

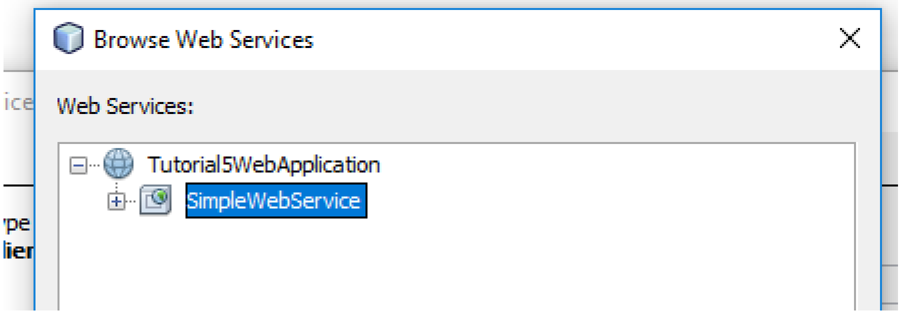
Package: [ ]

[ ] Generate Dispatch code

Enter the URL of the service you wish to use.

< Back | Next > | Finish | Cancel | Help

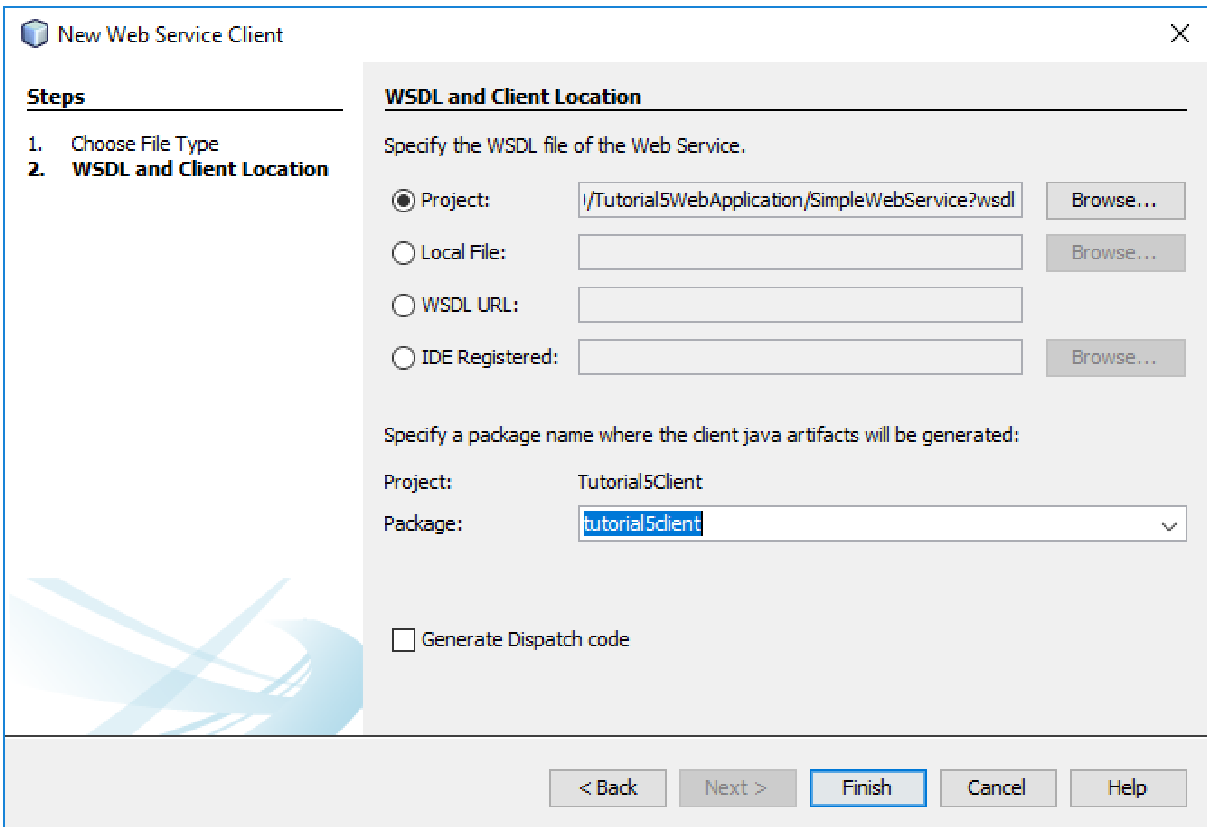
Figure 27,Creating the Web Service Client (Step 2)


Browse Web Services

Web Services:

Tutorial5WebApplication
SimpleWebService

Figure 28, Creating the Web Service Client (Step 3)


New Web Service Client

Steps

1. Choose File Type
2. **WSDL and Client Location**

WSDL and Client Location

Specify the WSDL file of the Web Service.

(●) Project: \/Tutorial5WebApplication/SimpleWebService?wsdl [Browse...]
( ) Local File: [Browse...]
( ) WSDL URL:
( ) IDE Registered: [Browse...]

Specify a package name where the client java artifacts will be generated:

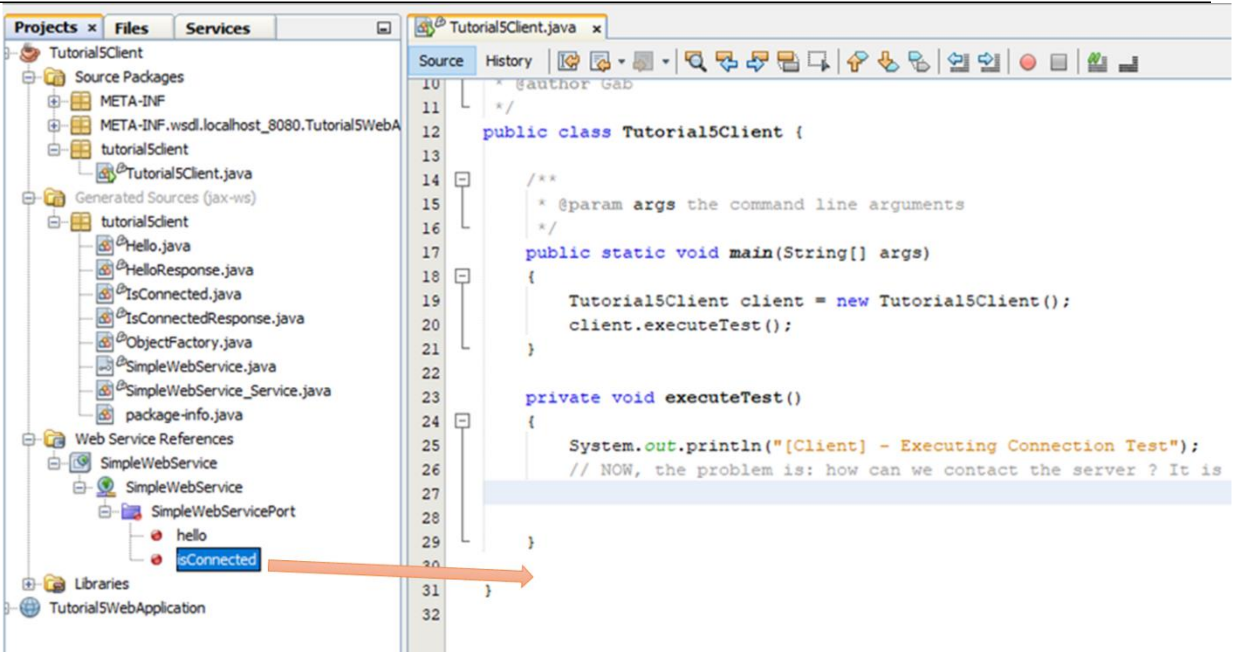
Project: Tutorial5Client
Package: tutorial5client

☐ Generate Dispatch code

< Back | Next > | Finish | Cancel | Help

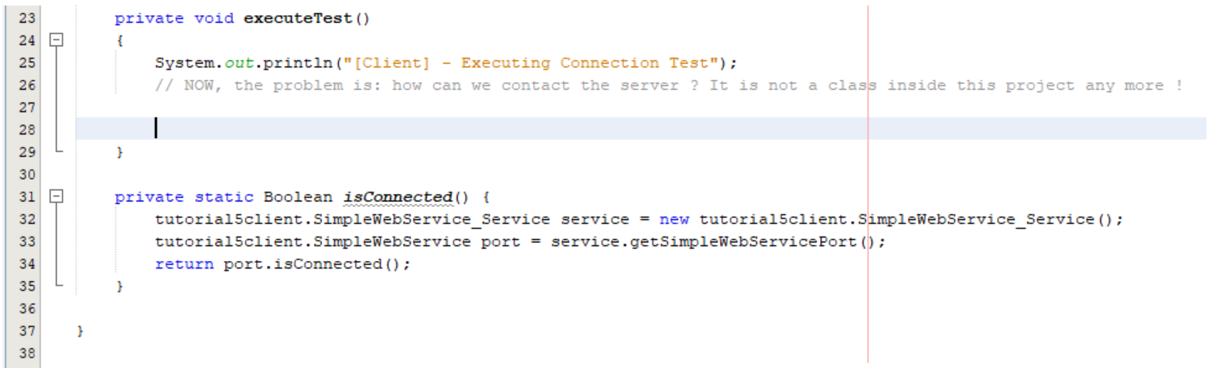
Figure 29, Creating the Web Service Client (Step 4)

18) You can notice that now on the client side, we have a representation of the server, 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 30 and Figure 31)



```
 * @author Gab
 */
public class Tutorial5Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        Tutorial5Client client = new Tutorial5Client();
        client.executeTest();
    }

    private void executeTest()
    {
        System.out.println("[Client] - Executing Connection Test");
        // NOW, the problem is: how can we contact the server ? It is


    }

}
```

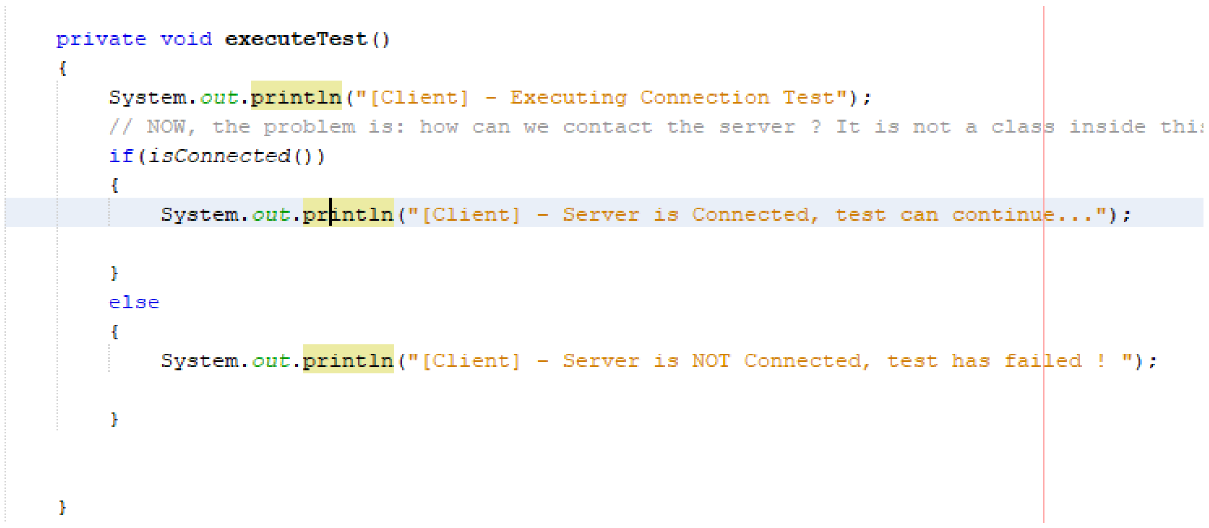
Figure 30, Creating Client Stubs (Step 1)



```
    private void executeTest()
    {
        System.out.println("[Client] - Executing Connection Test");
        // NOW, the problem is: how can we contact the server ? It is not a class inside this project any more !


    }

    private static Boolean isConnected() {
        tutorial5client.SimpleWebService_Service service = new tutorial5client.SimpleWebService_Service();
        tutorial5client.SimpleWebService port = service.getSimpleWebServicePort();
        return port.isConnected();
    }

}
```

Figure 31, Creating Client Stubs (Step 2)

19) Now you can invoke your client stub (which will in turn connect to the network and call the server) from your code as in Figure 32



```
private void executeTest()
{
    System.out.println("[Client] - Executing Connection Test");
    // NOW, the problem is: how can we contact the server ? It is not a class inside thi
    if(isConnected())
    {
        System.out.println("[Client] - Server is Connected, test can continue...");

    }
    else
    {
        System.out.println("[Client] - Server is NOT Connected, test has failed ! ");

    }


}
```

Figure 32, Invoking the client stub

20) There is one final step to complete before we can run the client, Netbeans has created a lot of classes that describe the server and how to contact it (you can take a look at them), we want to make a copy of them into our Java package to be able to use them (Figure 33). Use the copy – paste as refactor feature to do so (Figure 34, Figure 35, Figure 36 and Figure 37).

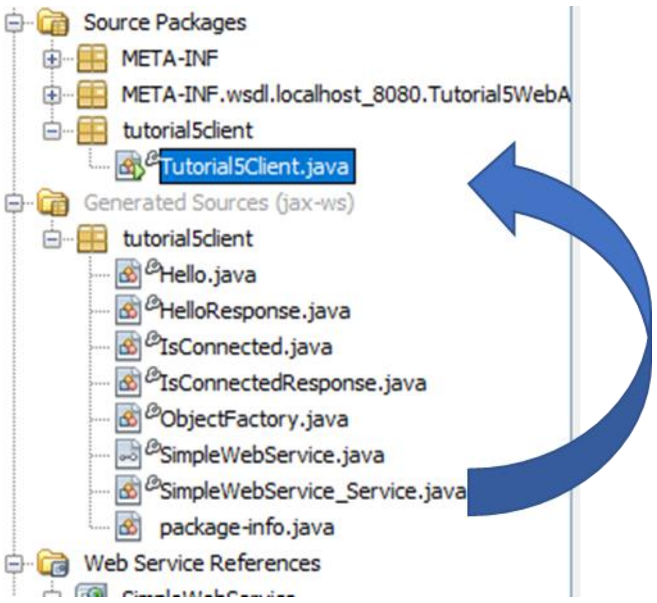


Figure 33, Automatically Generated Code for the client stubs

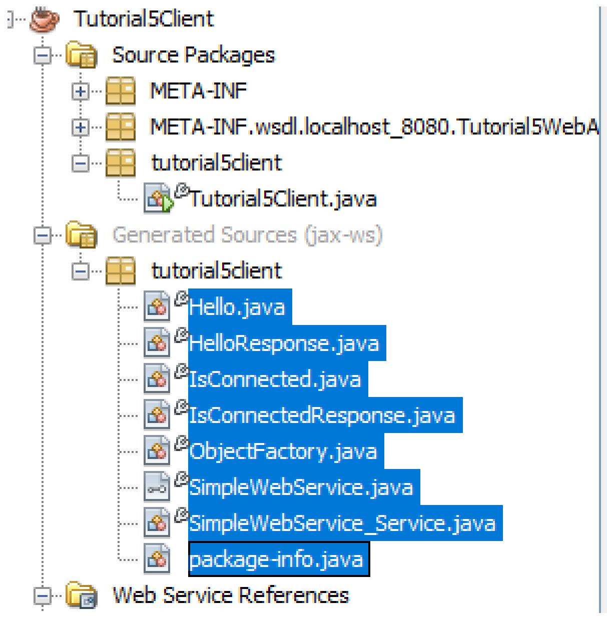


Figure 34, Refactor Copy of Generated code into the client package (step 1)

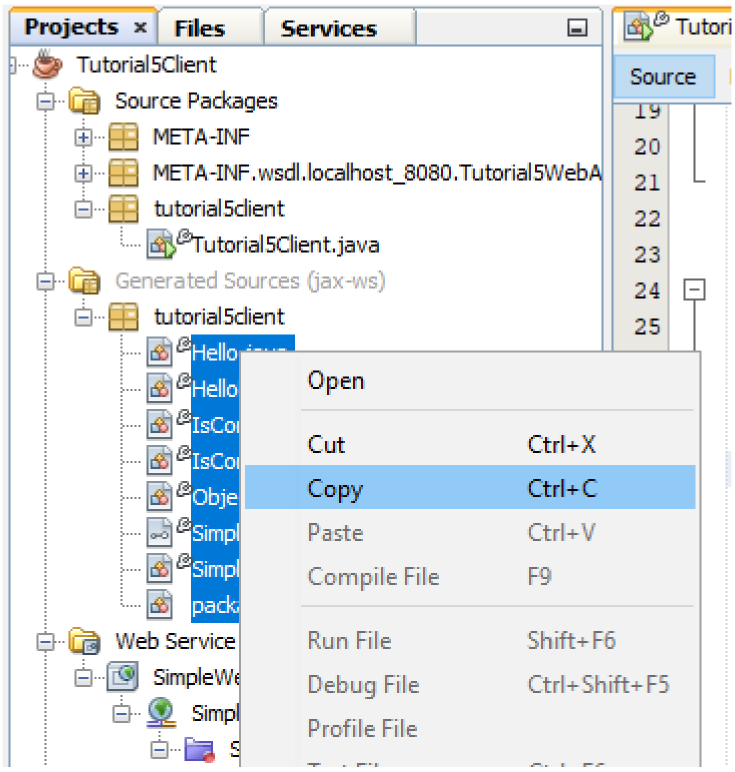


Figure 35, Refactor Copy of Generated code into the client package (step 2)

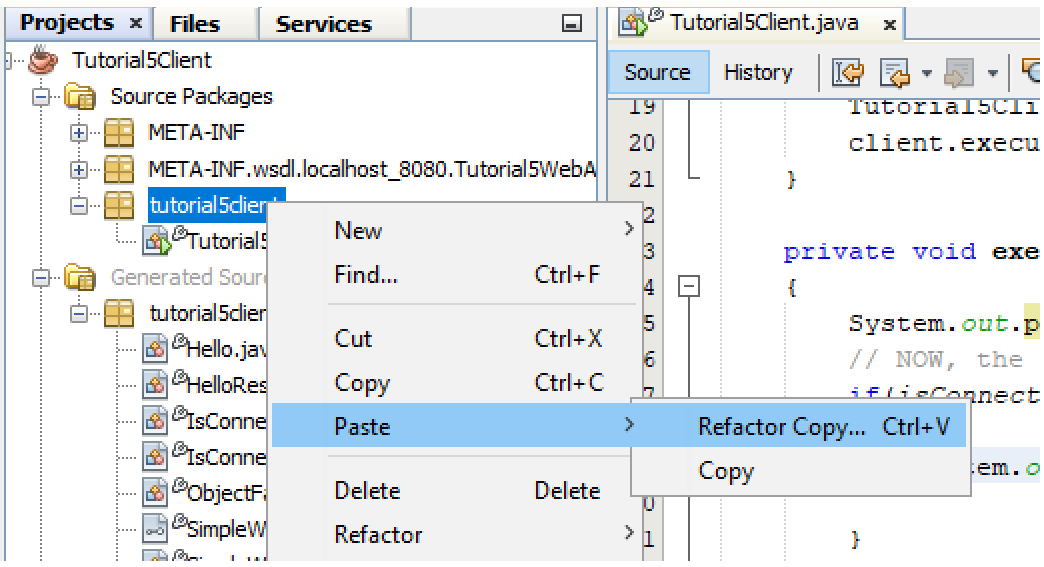
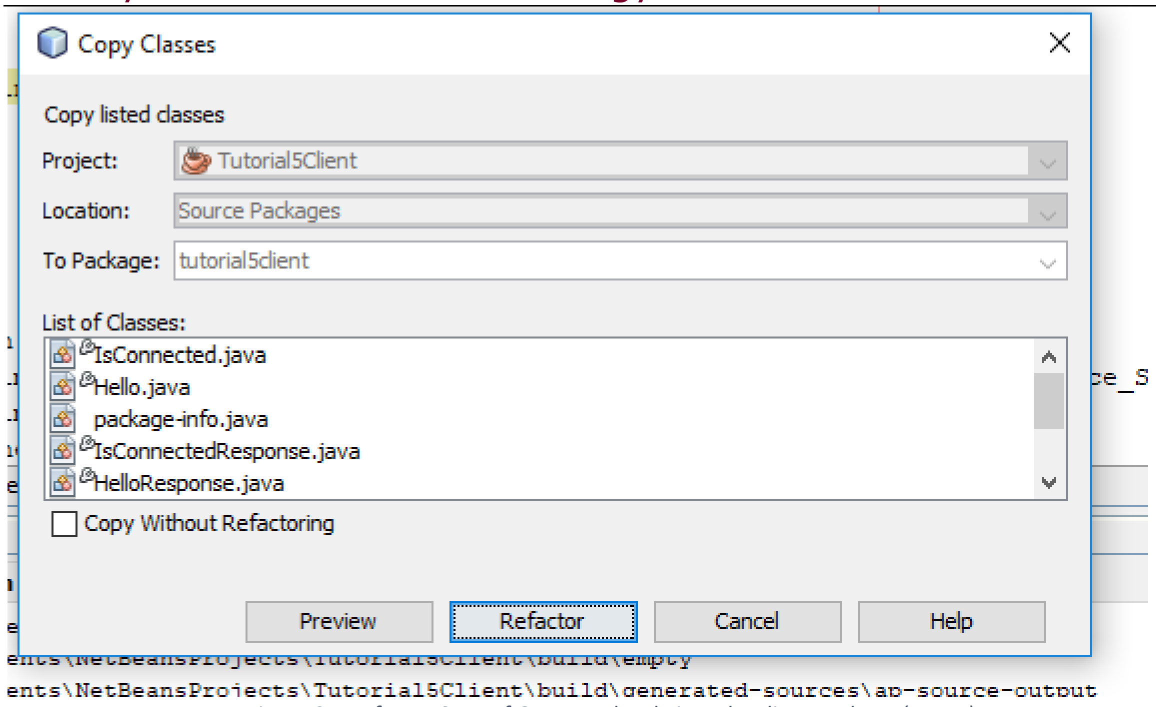


Figure 36, Refactor Copy of Generated code into the client package (step 3)



```
ents\NetBeansProjects\Tutorial5Client\build\empty
ents\NetBeansProjects\Tutorial5Client\build\generated-sources\ap-source-output
```

Figure 37, Refactor Copy of Generated code into the client package (step 4)

21) 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 (Figure 38).

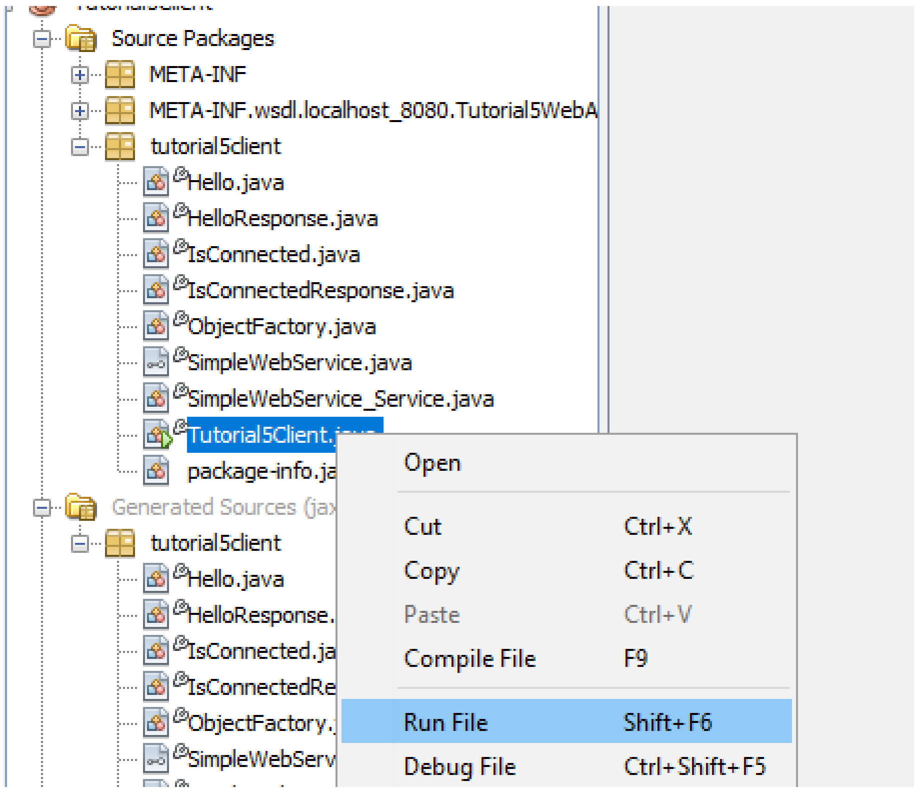
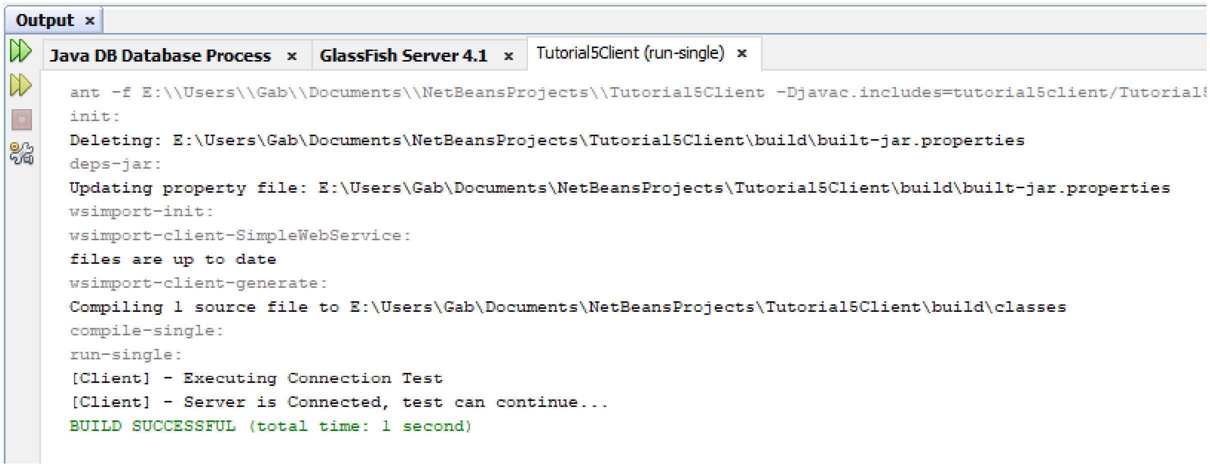


Figure 38, Test Client/Server Connection

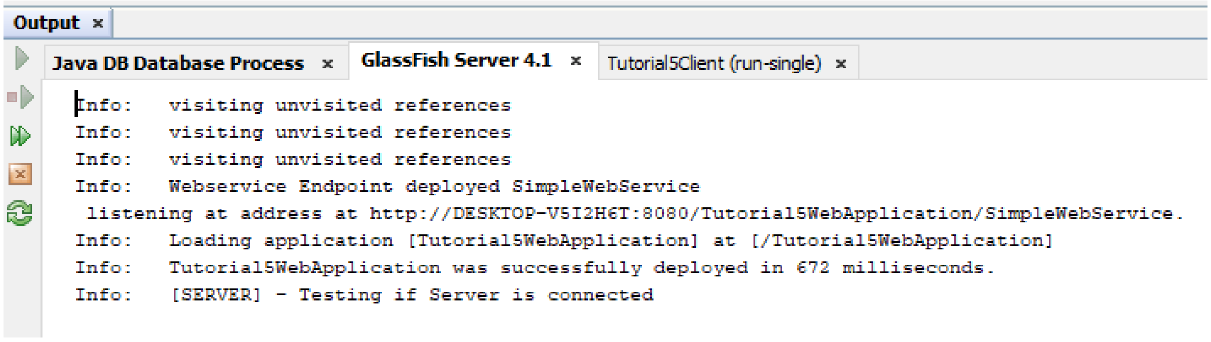
22) Observe how the logging calls (system.out.println) from the Server and the Client appear in different stdout streams (one from the GlassFish Server (Figure 40) and one from the Client (Figure 39).



Output ×
Java DB Database Process × | GlassFish Server 4.1 × | Tutorial5Client (run-single) ×

```
ant -f E:\\Users\\Gab\\Documents\\NetBeansProjects\\Tutorial5Client -Djavac.includes=tutorial5client/Tutorial5
init:
Deleting: E:\Users\Gab\Documents\NetBeansProjects\Tutorial5Client\build\built-jar.properties
deps-jar:
Updating property file: E:\Users\Gab\Documents\NetBeansProjects\Tutorial5Client\build\built-jar.properties
wsimport-init:
wsimport-client-SimpleWebService:
files are up to date
wsimport-client-generate:
Compiling 1 source file to E:\Users\Gab\Documents\NetBeansProjects\Tutorial5Client\build\classes
compile-single:
run-single:
[Client] - Executing Connection Test
[Client] - Server is Connected, test can continue...
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 39, Client Log



Output ×
Java DB Database Process × | GlassFish Server 4.1 × | Tutorial5Client (run-single) ×

```
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   Webservice Endpoint deployed SimpleWebService
 listening at address at http://DESKTOP-V5I2H6T:8080/Tutorial5WebApplication/SimpleWebService.
Info:   Loading application [Tutorial5WebApplication] at [/Tutorial5WebApplication]
Info:   Tutorial5WebApplication was successfully deployed in 672 milliseconds.
Info:   [SERVER] - Testing if Server is connected
```

Figure 40, Server Log

23) Export both the projects (client and server) as zip file on Netbeans.

TASKS to BE PERFORMED Independently be the student (from Task 24 to Task 28) (Formative Assessment)

24) Modify the Client so that it uses the hello method of the server to send a message from the client to the server.
25) Modify the server so that the server has a name (as a string) so that the server name is returned in the string from the method hello
26) Modify the server so that the server can add a time stamp to method hello (e.g. [Date and Time] - Server….. : Connection from client ….. succeded)
27) Investigate how you can obtain the IP address and hostnames of the client and the server and print them. At the moment client and server are separate programs but they run on the same machine
28) Investigate how you can modify the client stubs to try to connect to a server on another remote machine and try it !