University of Westminster
Faculty of Science and Technology

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

Tutorial title	**Introduction to Web Services**
Tutorial type	**Guided and indepenent and non-marked**
Week 05	**04/02/2021**

Contents

Learning Goals

This tutorial focuses on a fundamental new learning goal to create a REAL Client/Server system based on the Web Service Technology.
While the previous tutorials (1 and 2) 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 and 2 and Tutorial 3 Architectures).

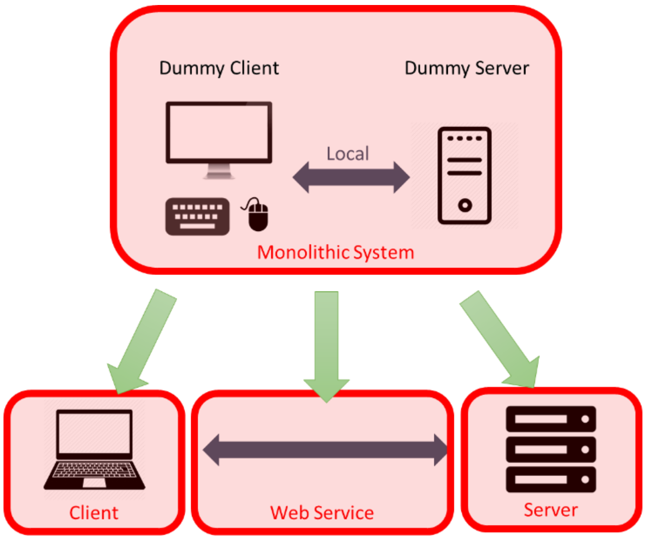
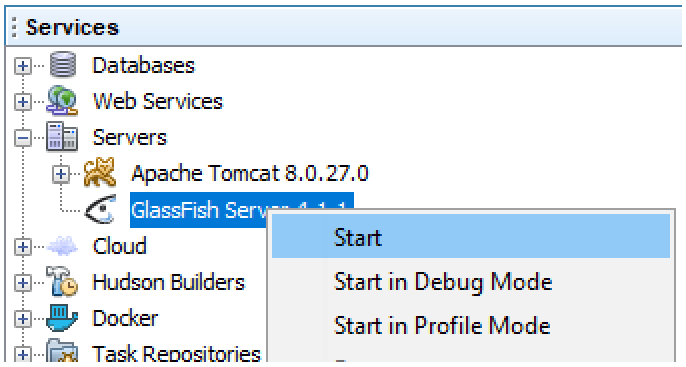
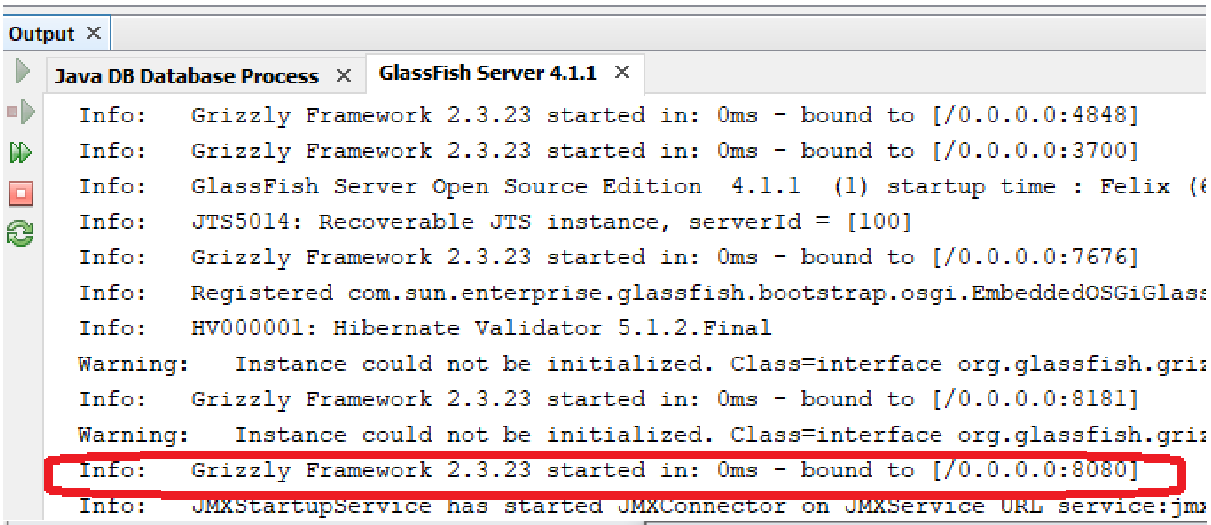


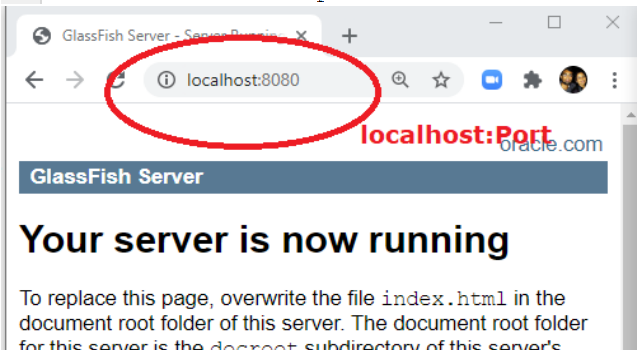
Figure 1, Tutorial 1 and 2 and Tutorial 3 Architectures

TEST IF Glassfish Works - your installation may boot on a different port.
Start glassfish, check if the home page loads up, then stop it before you do this tutorial. If it does not work then check with tutor.



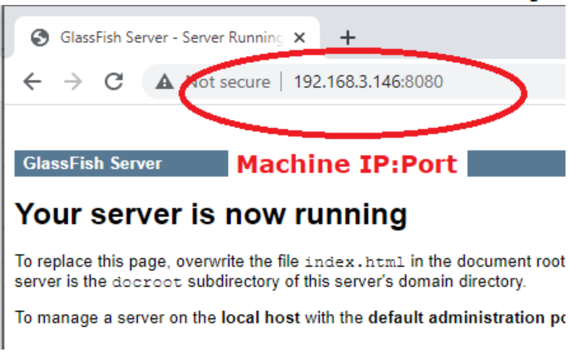


```
Info:   Grizzly Framework 2.3.23 started in: 0ms - bound to [/0.0.0.0:4848]
Info:   Grizzly Framework 2.3.23 started in: 0ms - bound to [/0.0.0.0:3700]
Info:   GlassFish Server Open Source Edition  4.1.1  (1) startup time : Felix (
Info:   JTS5014: Recoverable JTS instance, serverId = [100]
Info:   Grizzly Framework 2.3.23 started in: 0ms - bound to [/0.0.0.0:7676]
Info:   Registered com.sun.enterprise.glassfish.bootstrap.osgi.EmbeddedOSGiGlass
Info:   HV000001: Hibernate Validator 5.1.2.Final
Warning:   Instance could not be initialized. Class=interface org.glassfish.gri
Info:   Grizzly Framework 2.3.23 started in: 0ms - bound to [/0.0.0.0:8181]
Warning:   Instance could not be initialized. Class=interface org.glassfish.gri
Info:   Grizzly Framework 2.3.23 started in: 0ms - bound to [/0.0.0.0:8080]
Info:   JMXStartupService has started JMXConnector on JMXService URL service:jmx
```



Your server is now running

To replace this page, overwrite the file `index.html` in the document root folder of this server. The document root folder for this server is the `docroot` subdirectory of this server's

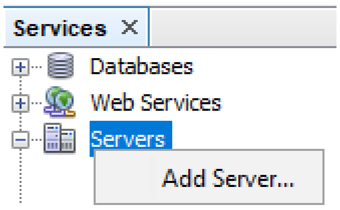

GlassFish Server - Server Running
Not secure | 192.168.3.146:8080
GlassFish Server
Machine IP:Port

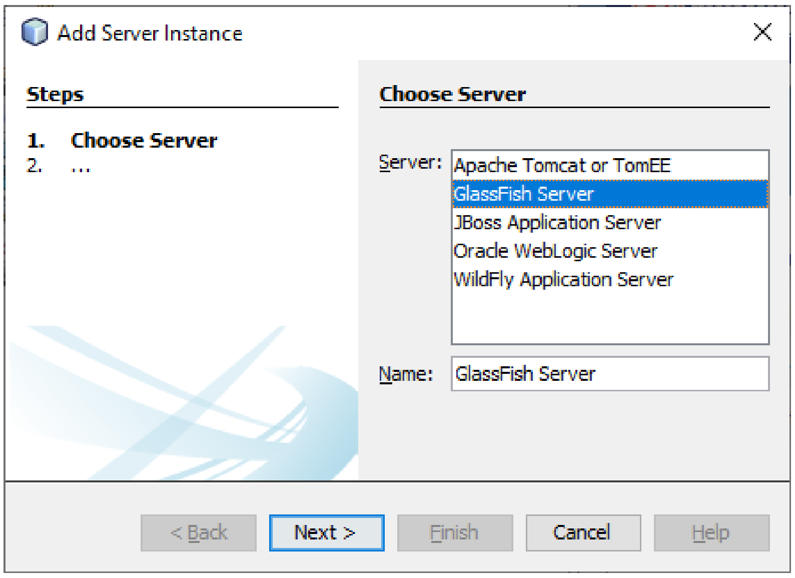
Your server is now running

To replace this page, overwrite the file `index.html` in the document root server is the `docroot` subdirectory of this server's domain directory.

To manage a server on the **local host** with the **default administration p**

IF No GlassFish, try the following to make Netbeans show it.



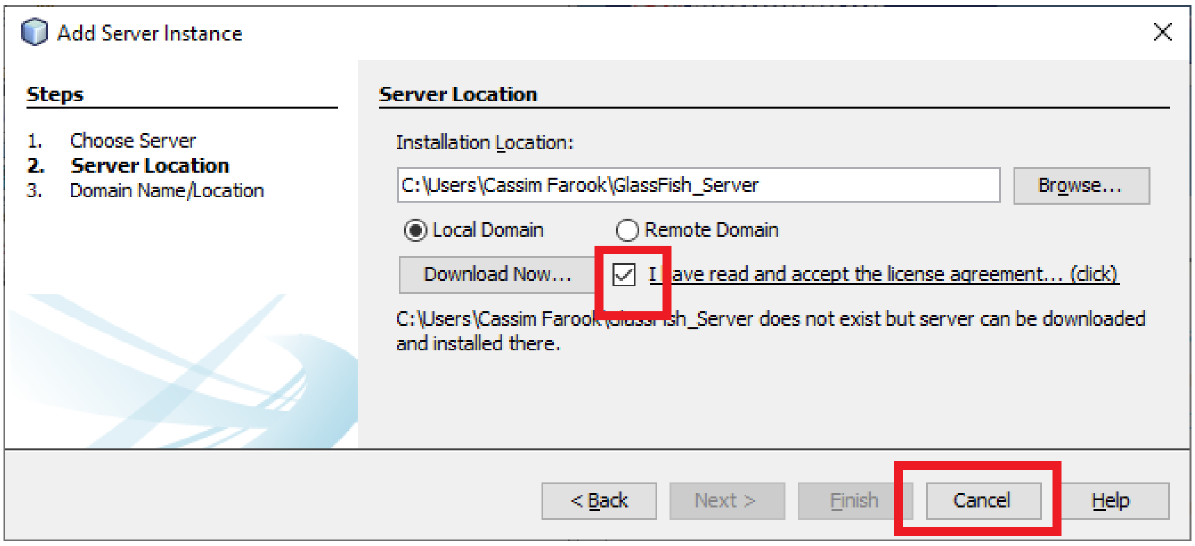


Server: Apache Tomcat or TomEE / GlassFish Server / JBoss Application Server / Oracle WebLogic Server / WildFly Application Server

Name: GlassFish Server

This might work if you have the full installation.

On second screen, tick and press "Cancel"



Installation Location: C:\Users\Cassim Farook\GlassFish_Server

[✓ Local Domain] Remote Domain

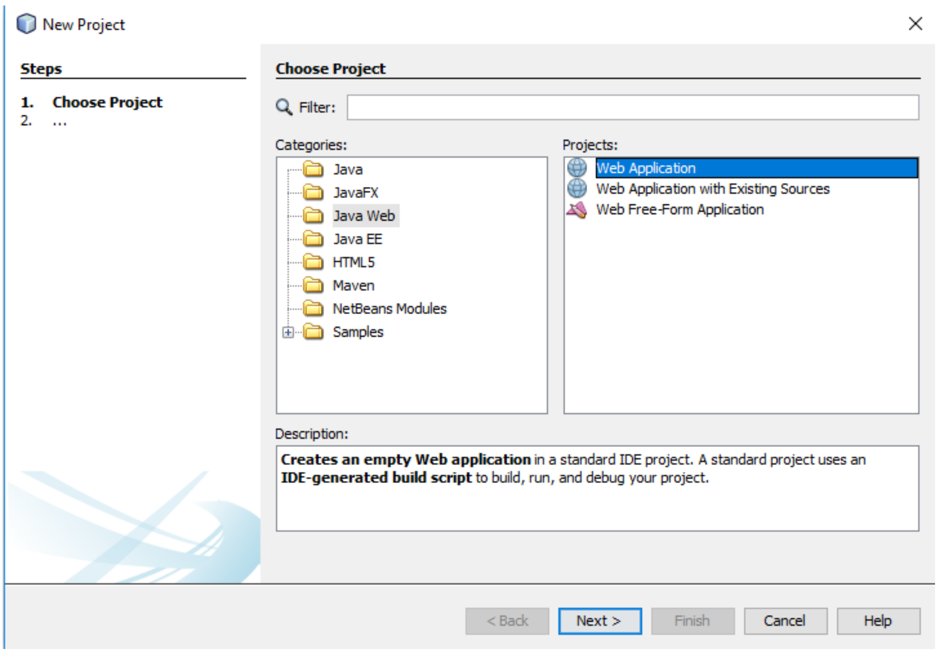
☑ I have read and accept the license agreement... (click)

C:\Users\Cassim Farook\GlassFish_Server does not exist but server can be downloaded and installed there.

As usual, 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)

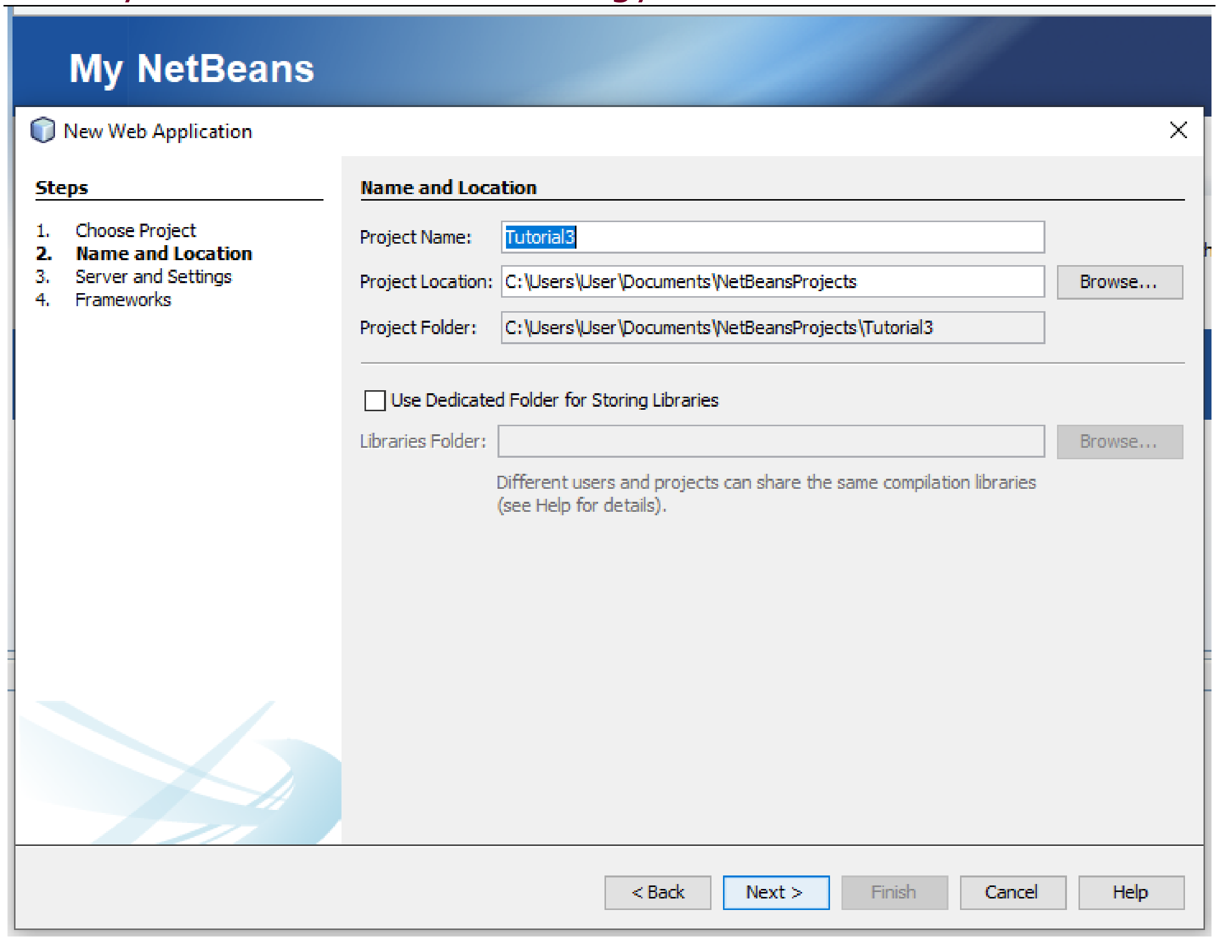
1) Start Netbeans in your system.
2) Instead of creating a new java project, this time we create a new Web Application Project (Figure 2, Create new Web Application Project (Step 1)



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 2, Create new Web Application Project (Step 1)

3) Let’s call our first Web Application Tutorial3



Name and Location

Project Name: Tutorial3

Project Location: C:\Users\User\Documents\NetBeansProjects

Project Folder: C:\Users\User\Documents\NetBeansProjects\Tutorial3

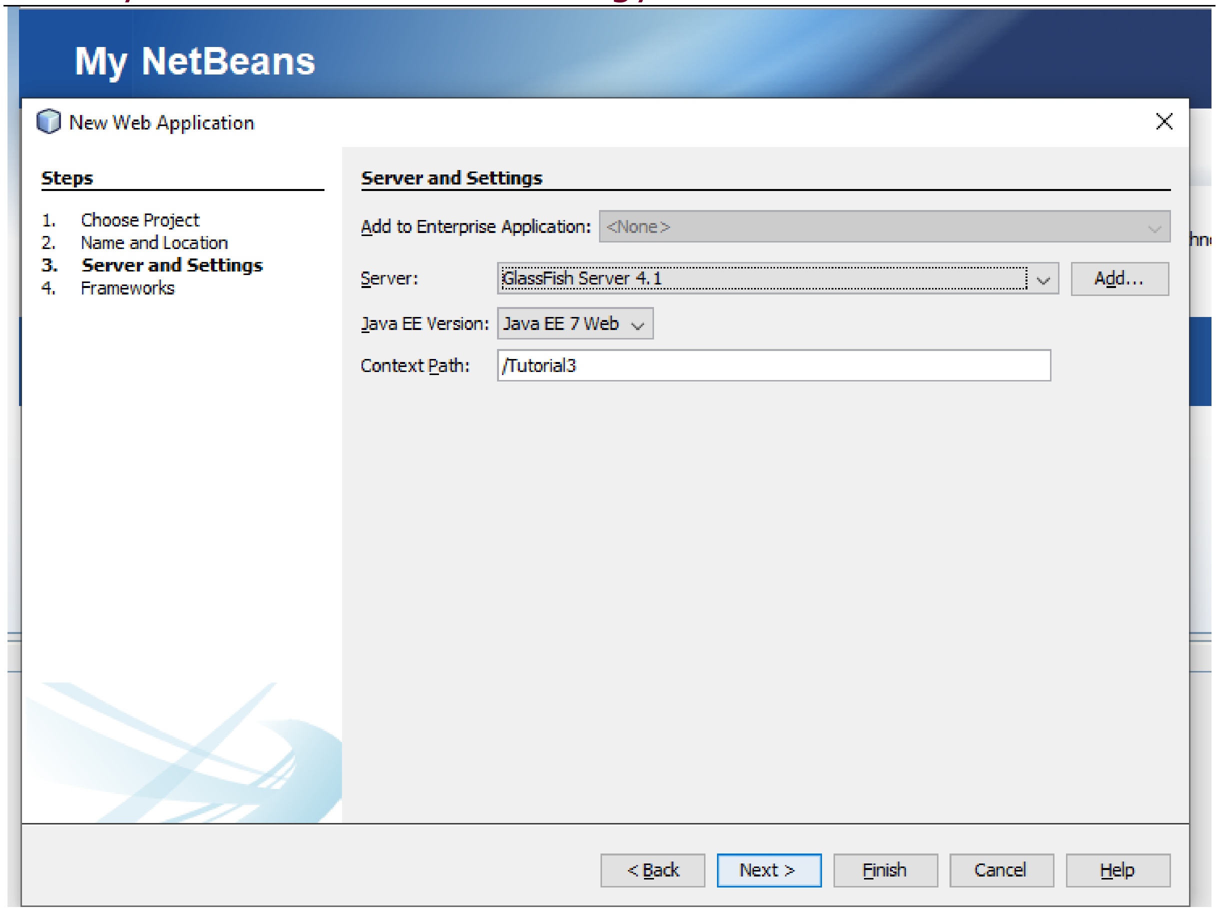
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

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

Figure 3,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 4,Create new Web Application Project (Step 3))


My NetBeans

New Web Application

Steps

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

Server and Settings

Add to Enterprise Application: <None>

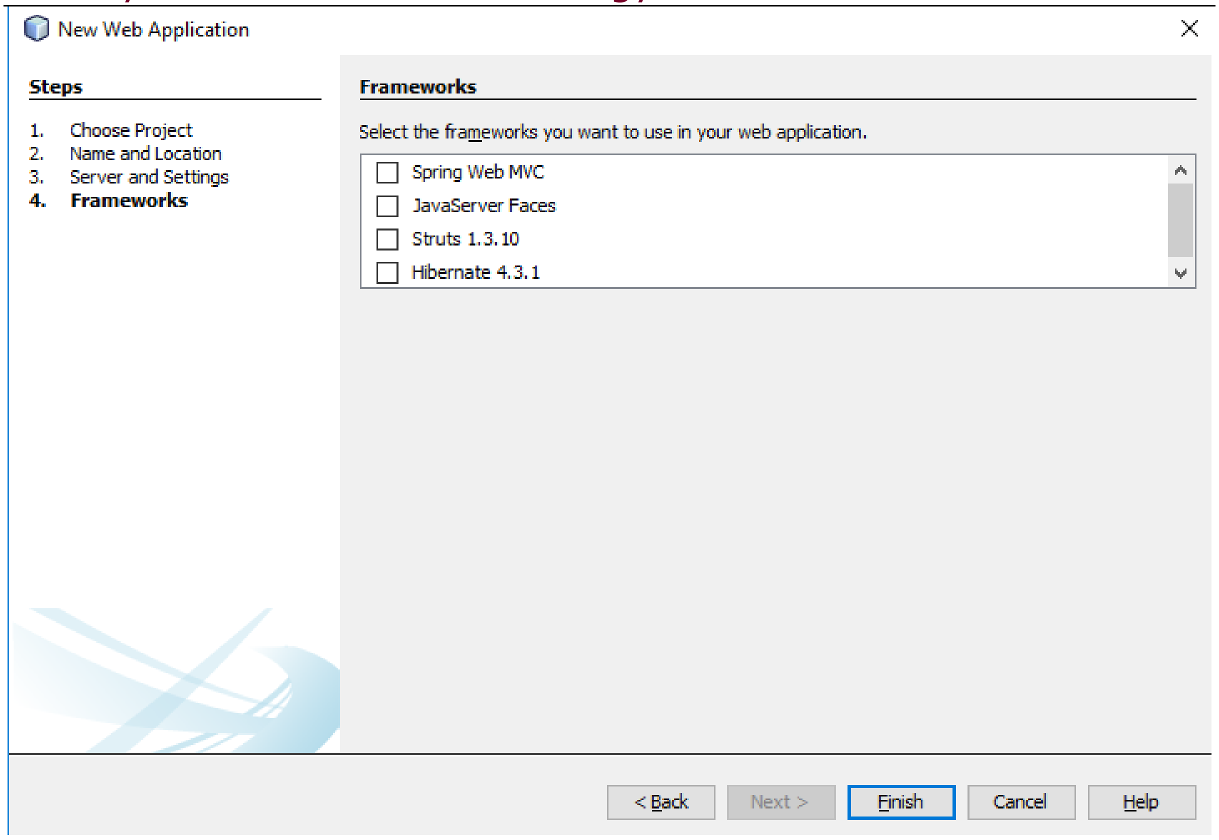
Server: GlassFish Server 4.1

Java EE Version: Java EE 7 Web

Context Path: /Tutorial3

Figure 4,Create new Web Application Project (Step 3)

5) Do not select any framework. (Figure 5, Create new Web Application Project (Step 4))

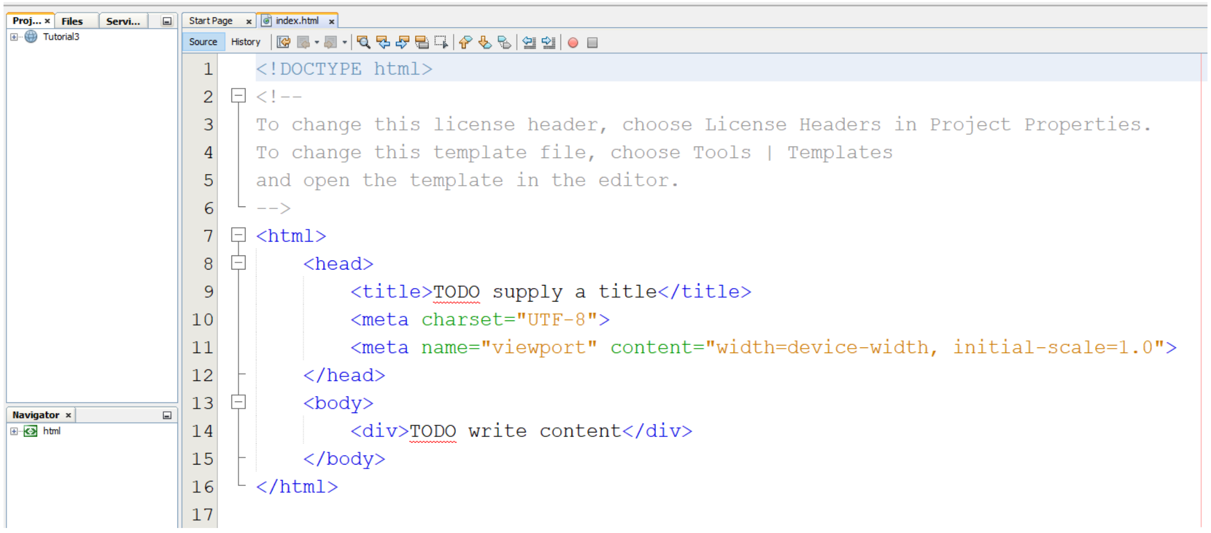


Select the frameworks you want to use in your web application.

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

Figure 5, 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 6, Create new Web Application Project (Step 5)).



```
<!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 6, 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 7, Running the new Web Application Project (Step 1) and Figure 8, Running the new Web

Application Project (Step 2). Our Web Application at the moment is just a simple Web Page.

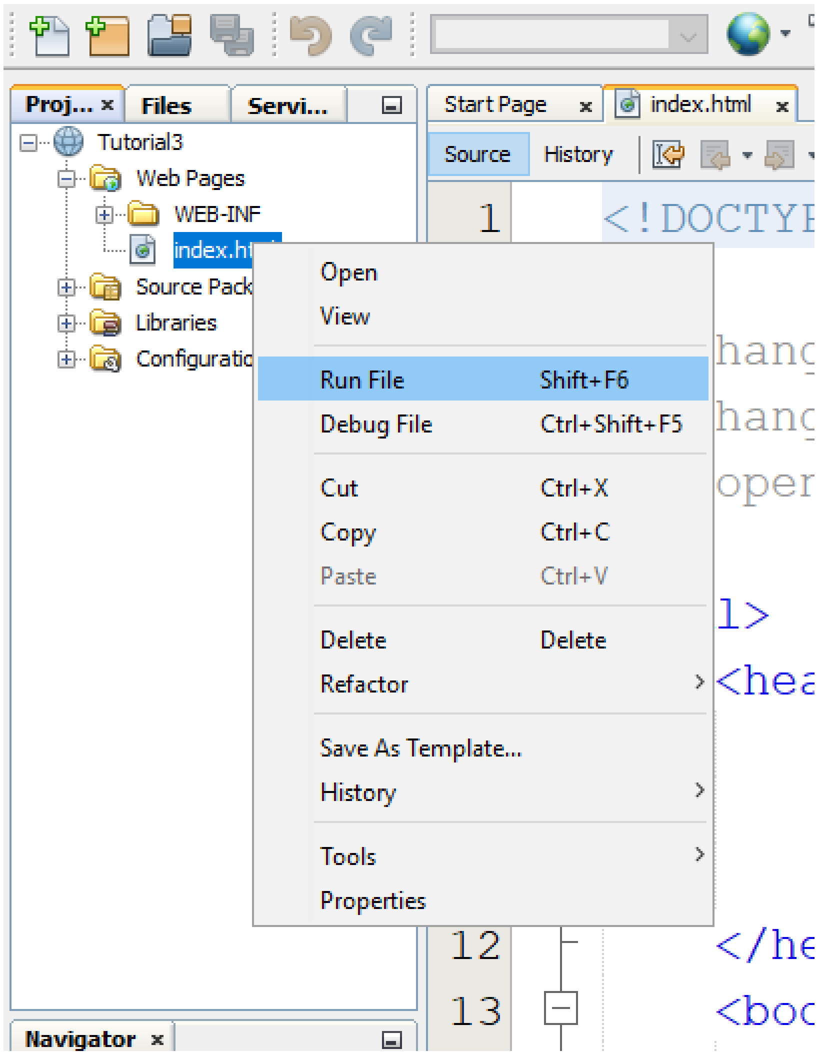
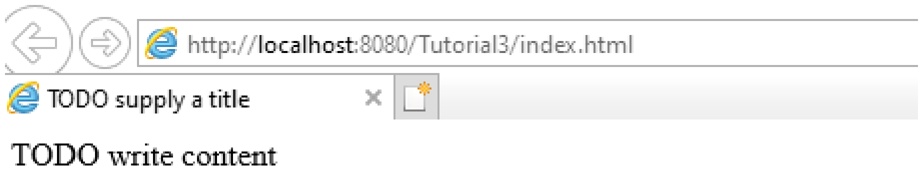


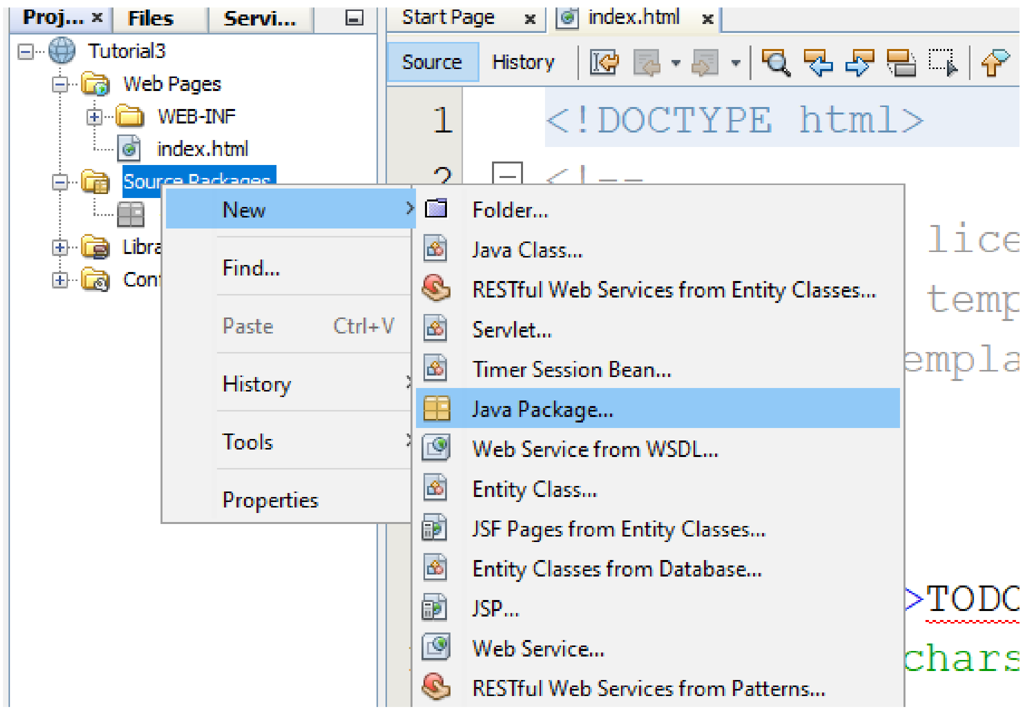
Figure 7, Running the new Web Application Project (Step 1)



TODO write content

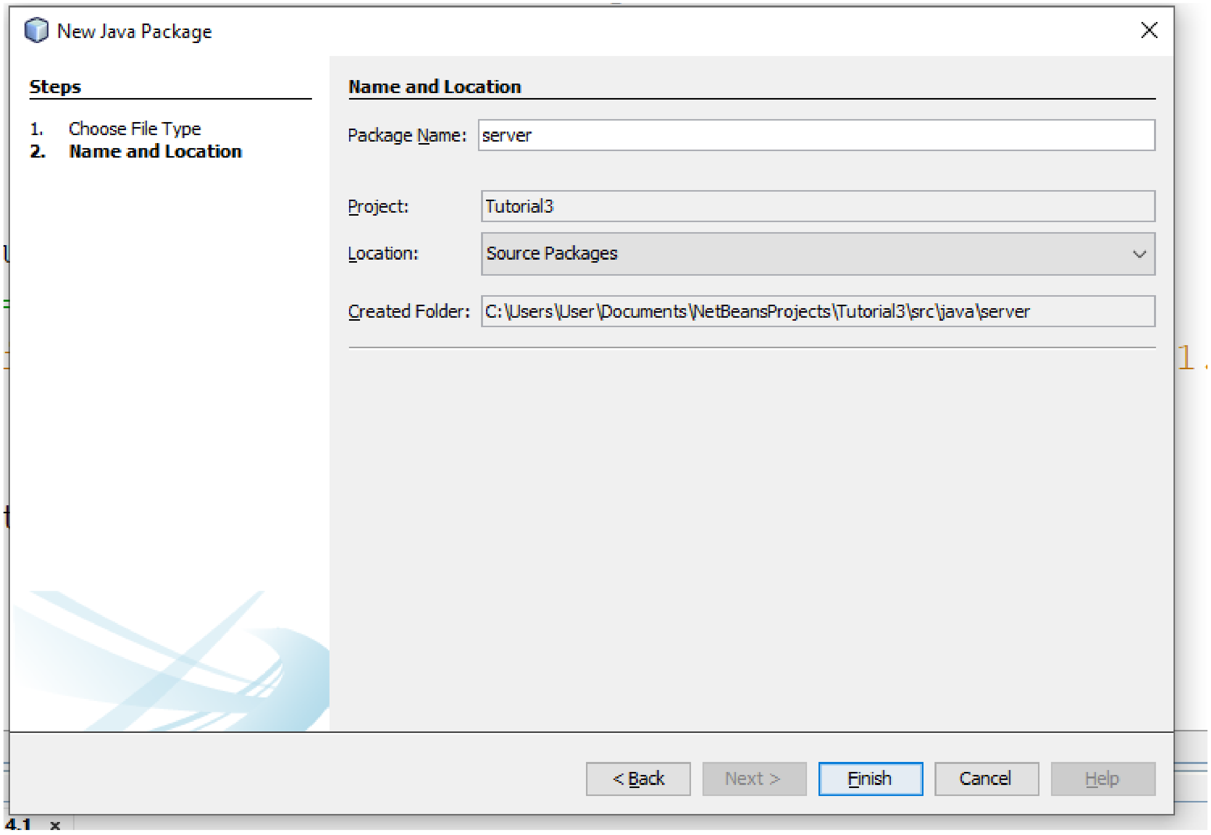
Figure 8, Running the new Web Application Project (Step 2)

8) Create a new Java package in the Web Application where we will put our server, called it server (Figure 9, Creating a server Java Package and Figure 10,Creating a server Java Package (Step 2))



```
<!DOCTYPE html>
```

Figure 9, Creating a server Java Package (Step 1)



Package Name: server

Project: Tutorial3

Location: Source Packages

Created Folder: C:\Users\User\Documents\NetBeansProjects\Tutorial3\src\java\server

Figure 10,Creating a server Java Package (Step 2)

9) Now, we can create a proper Web Service, we will call it SimpleWebService (Figure 11, Create the Web Service (step 1) and Figure 12,Create the Web Service (step 2))

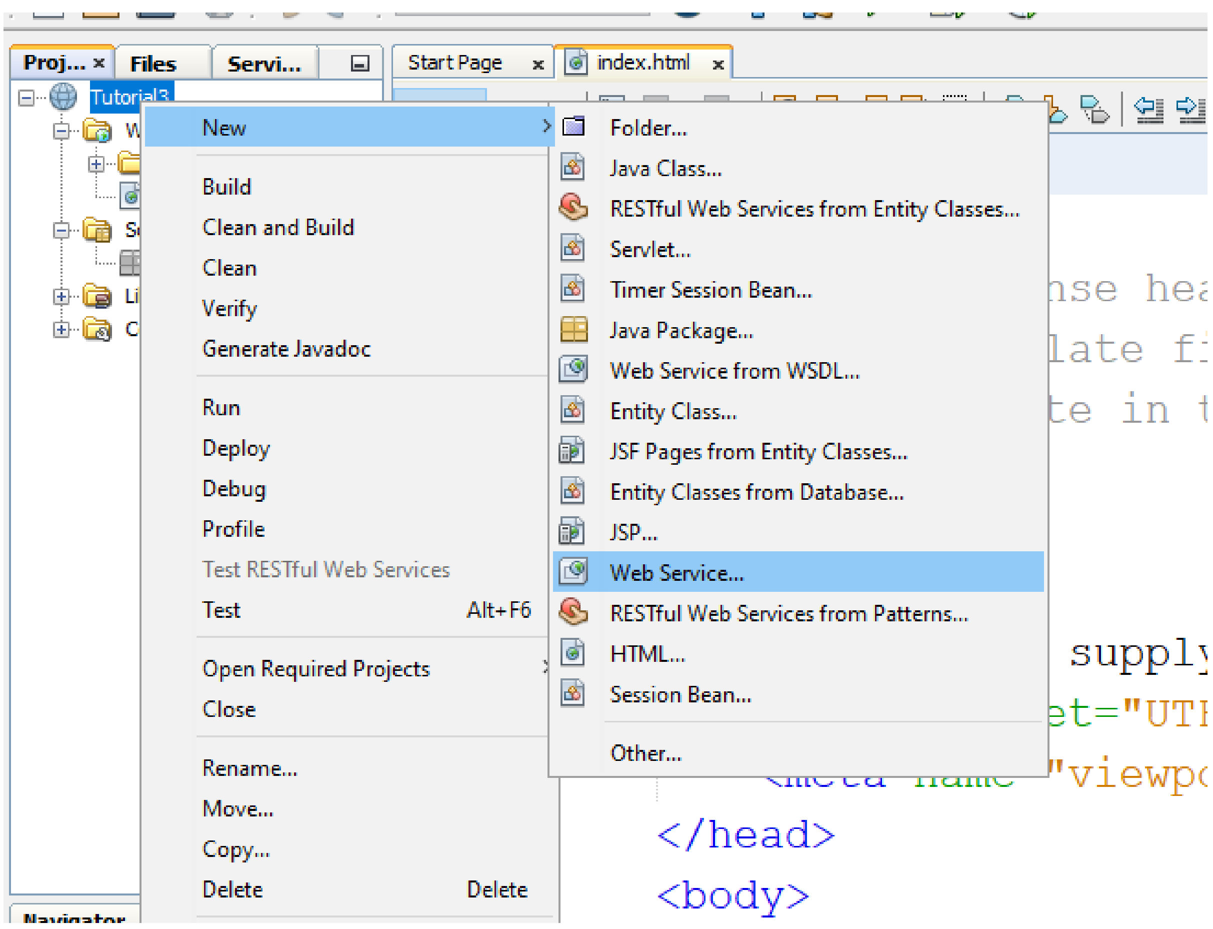
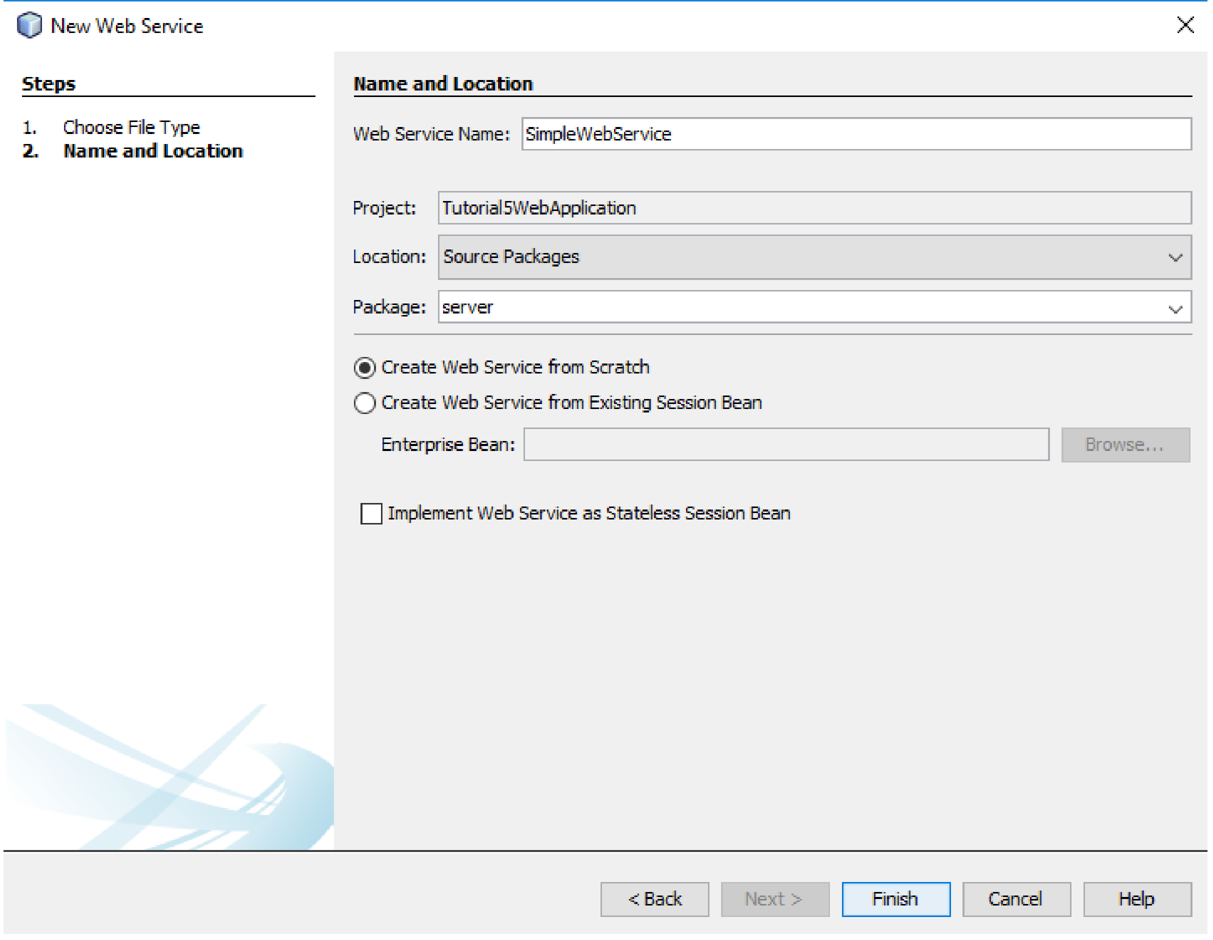


Figure 11, Create the Web Service (step 1)


New Web Service

Steps

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

Name and Location

Web Service Name: SimpleWebService

Project: Tutorial5WebApplication

Location: Source Packages

Package: server

(●) Create Web Service from Scratch

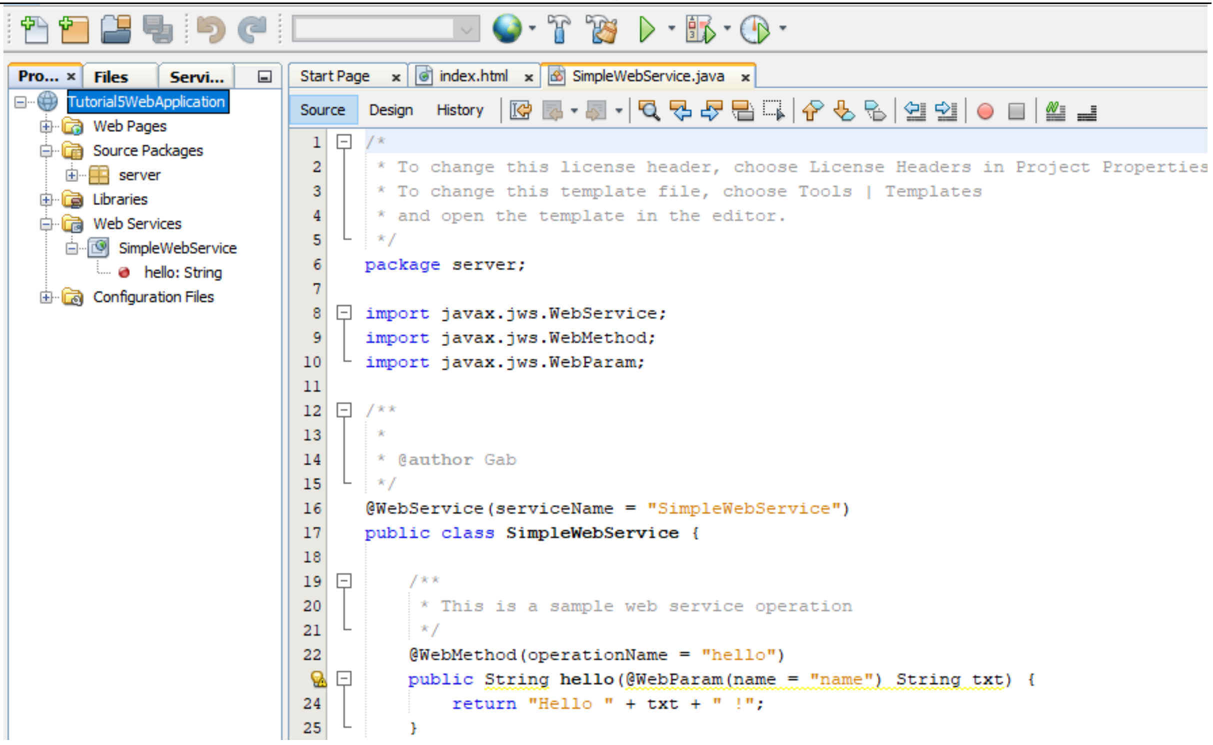
( ) Create Web Service from Existing Session Bean

Enterprise Bean:

[ ] Implement Web Service as Stateless Session Bean

Figure 12,Create the Web Service (step 2)

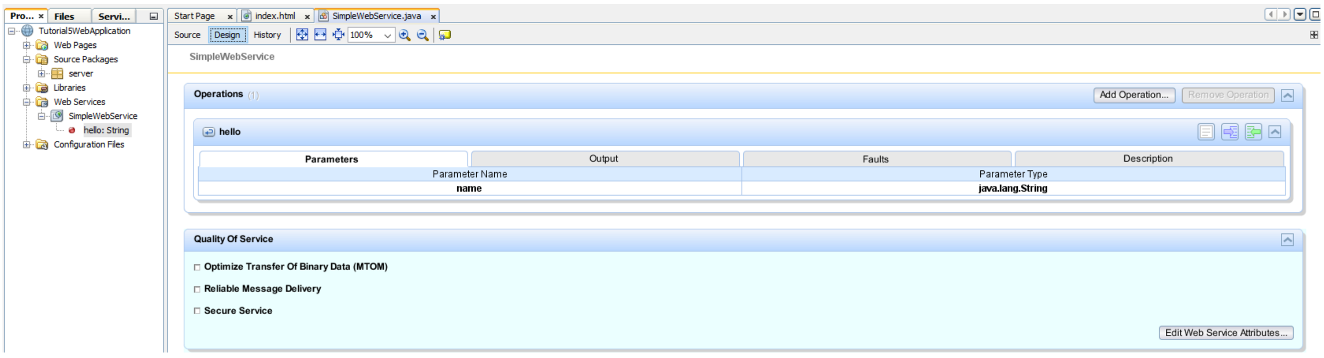
10) You can observe that NetBeans creates a standar method called Hello which echoes the message sent. (Figure 13, Standard Web Service Created by Netbeans.)



```
/*
 * 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 13, 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 14, Design view of the Web Service).

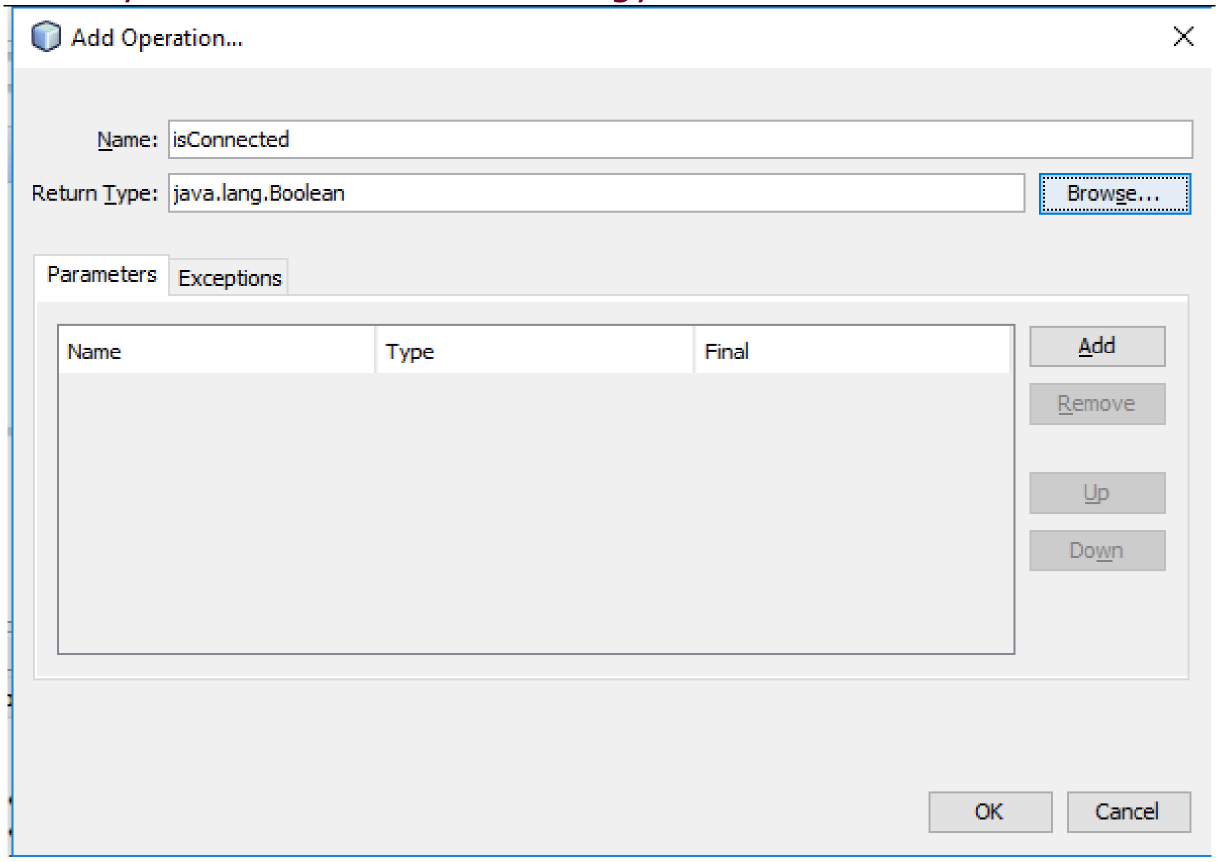


Parameter Name	Parameter Type
name	java.lang.String

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

Figure 14, 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 15, Adding the isConnected method to the Server (step 1) and Figure 16, Adding the isConnected method to the Server (step 2)).


Add Operation...

Name: isConnected

Return Type: java.lang.Boolean

Figure 15, Adding the isConnected method to the Server (step 1)

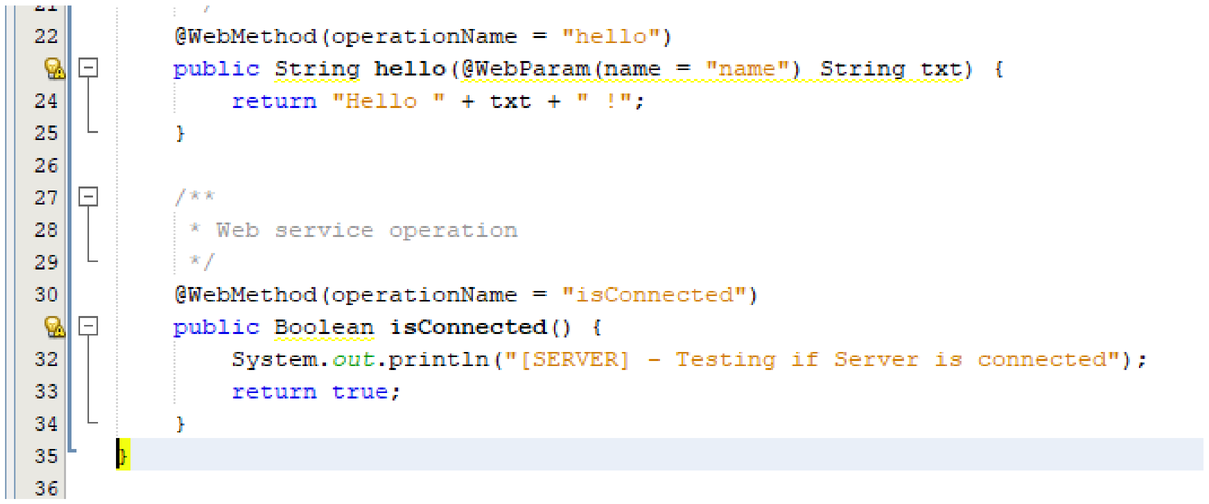
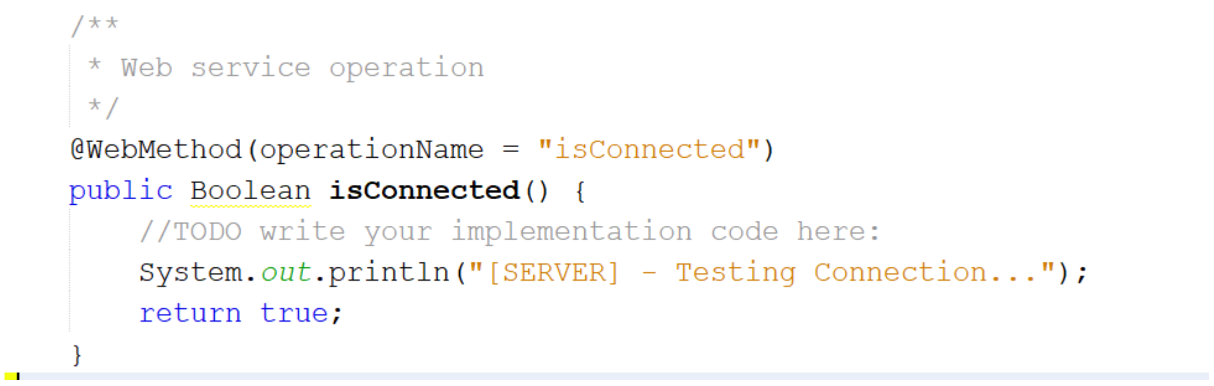

```
    @WebMethod(operationName = "hello")
    public String hello(@WebParam(name = "name") String txt) {
        return "Hello " + txt + " !";
    }

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

Figure 16, Adding the isConnected method to the Server (step 2)

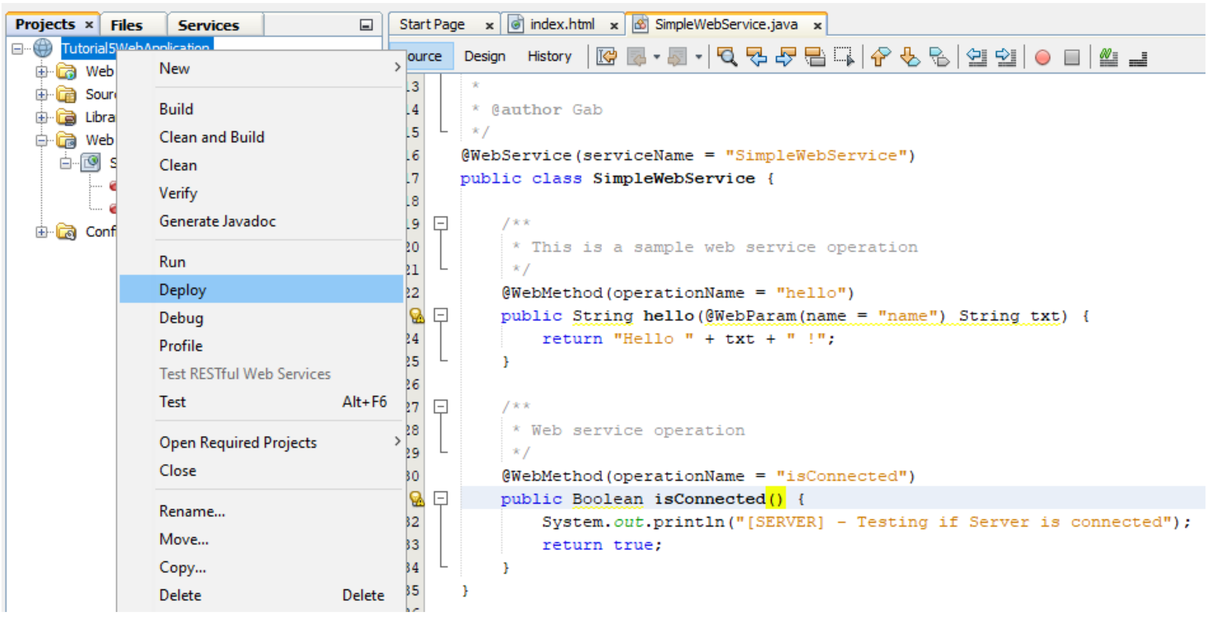
13) Add the usual simple logic of the usual isConnected method in the logic of the Web Service (Figure 17, Logic on the Web Service Method isConnected.)



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

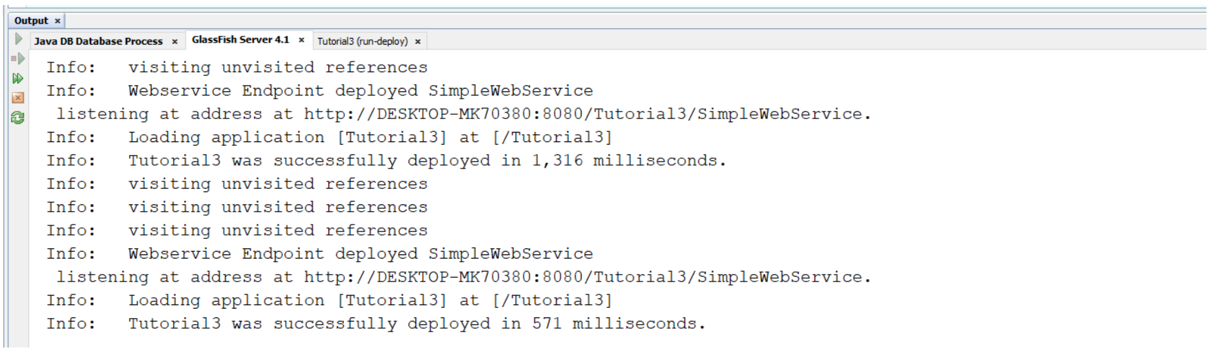
Figure 17, Logic on the Web Service Method isConnected.

14) Now we deploy the server on the glassfish server engine (Figure 18, Deploying the Server). You can observe the server logs (Figure 19, Server Log). **ALWAYS REMEMBER TO REDEPLOY THE SERVER EVERY TIME YOU CHANGE ITS CODE.**



```
 *
 * @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



```
Info:   visiting unvisited references
Info:   Webservice Endpoint deployed SimpleWebService
 listening at address at http://DESKTOP-MK70380:8080/Tutorial3/SimpleWebService.
Info:   Loading application [Tutorial3] at [/Tutorial3]
Info:   Tutorial3 was successfully deployed in 1,316 milliseconds.
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   visiting unvisited references
Info:   Webservice Endpoint deployed SimpleWebService
 listening at address at http://DESKTOP-MK70380:8080/Tutorial3/SimpleWebService.
Info:   Loading application [Tutorial3] at [/Tutorial3]
Info:   Tutorial3 was successfully deployed in 571 milliseconds.
```

Figure 19, Server Log

15) NetBeans offers us a testing tool without even to have to write a client (NetBeans will write a simple client in a browser)(Figure 20, Testing the Web Service (Step 1)). Test the isConnected method, it returns true, that is correct (Figure 21,Testing the Web Service (Step 2) and Figure 22, Testing the Web Service (Step 3))! You can also check the server log to see if it is correct (Figure 23, Test Server Log)

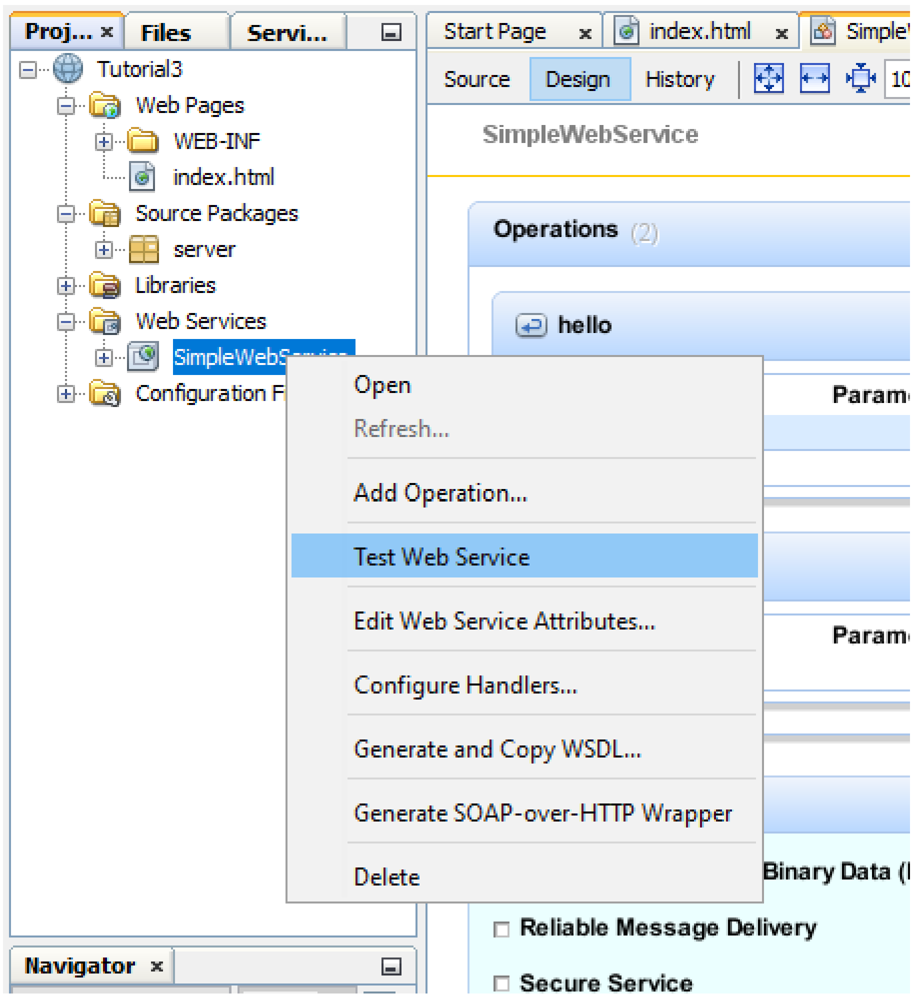
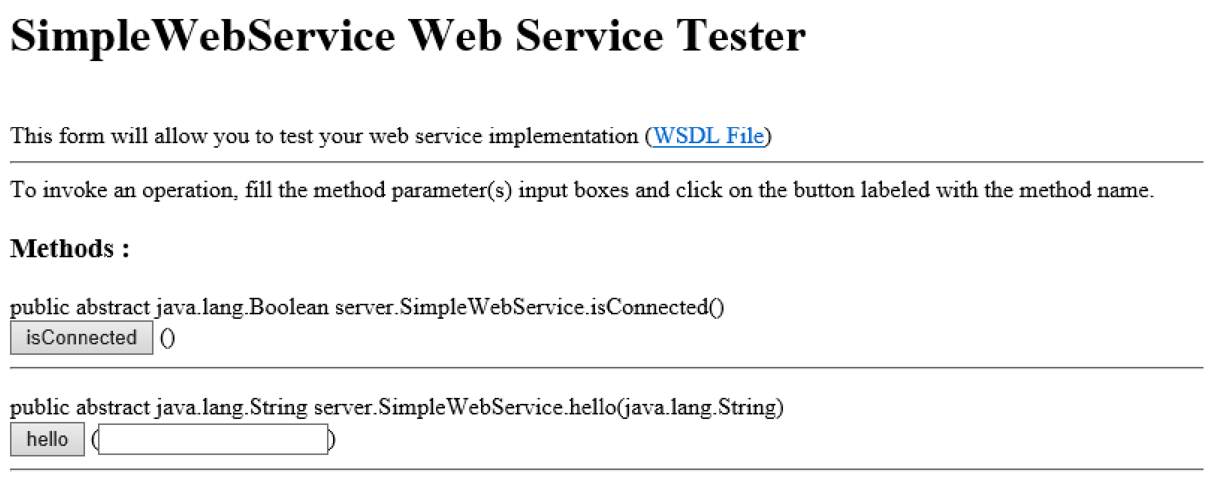


Figure 20, Testing the Web Service (Step 1)



SimpleWebService Web Service Tester

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

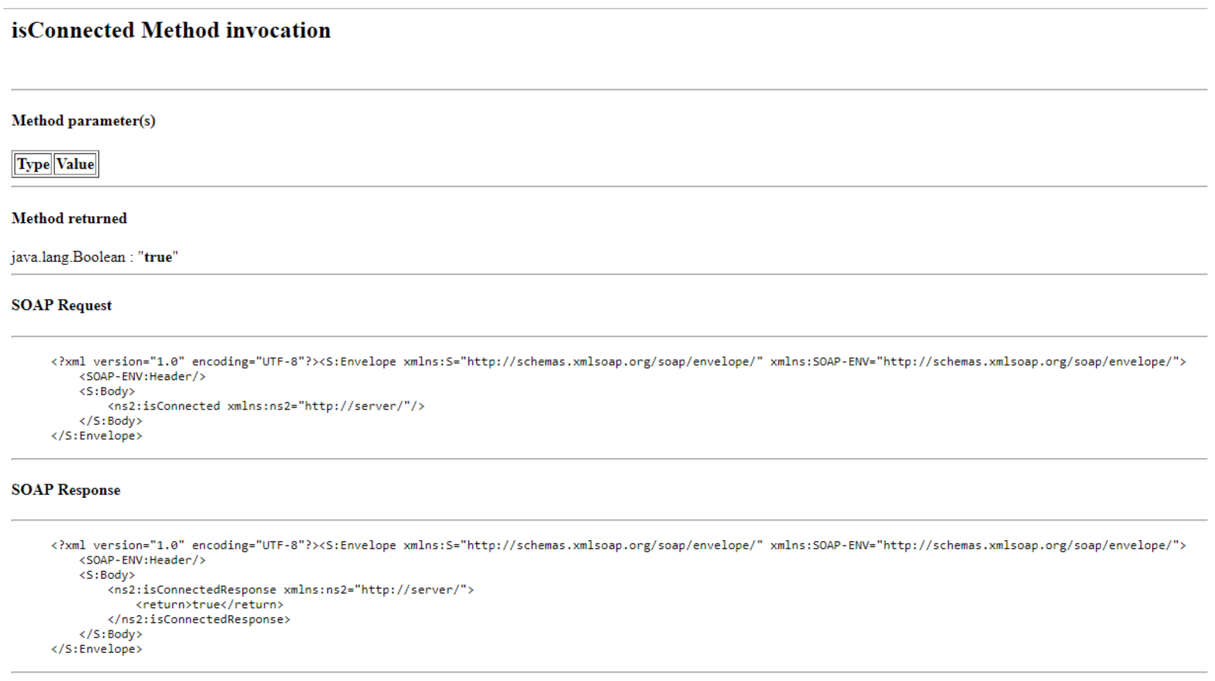
To invoke an operation, fill the method parameter(s) input boxes and click on the button labeled with the method name.

Methods :

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

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

Figure 21,Testing the Web Service (Step 2)



isConnected Method invocation

Method parameter(s)

Type	Value

Method returned

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

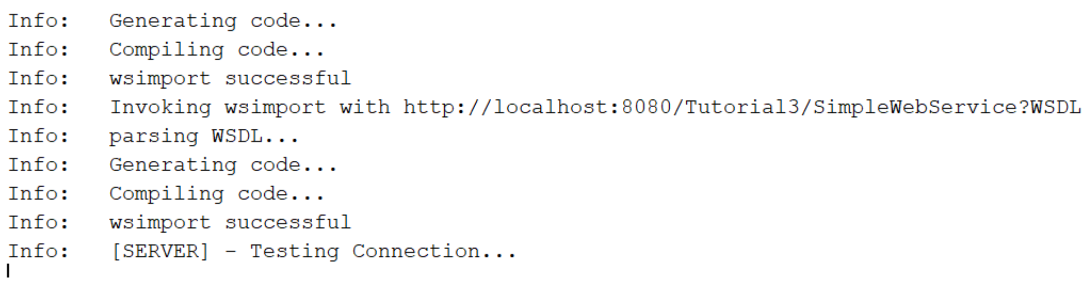
SOAP Request

```
<?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>
```

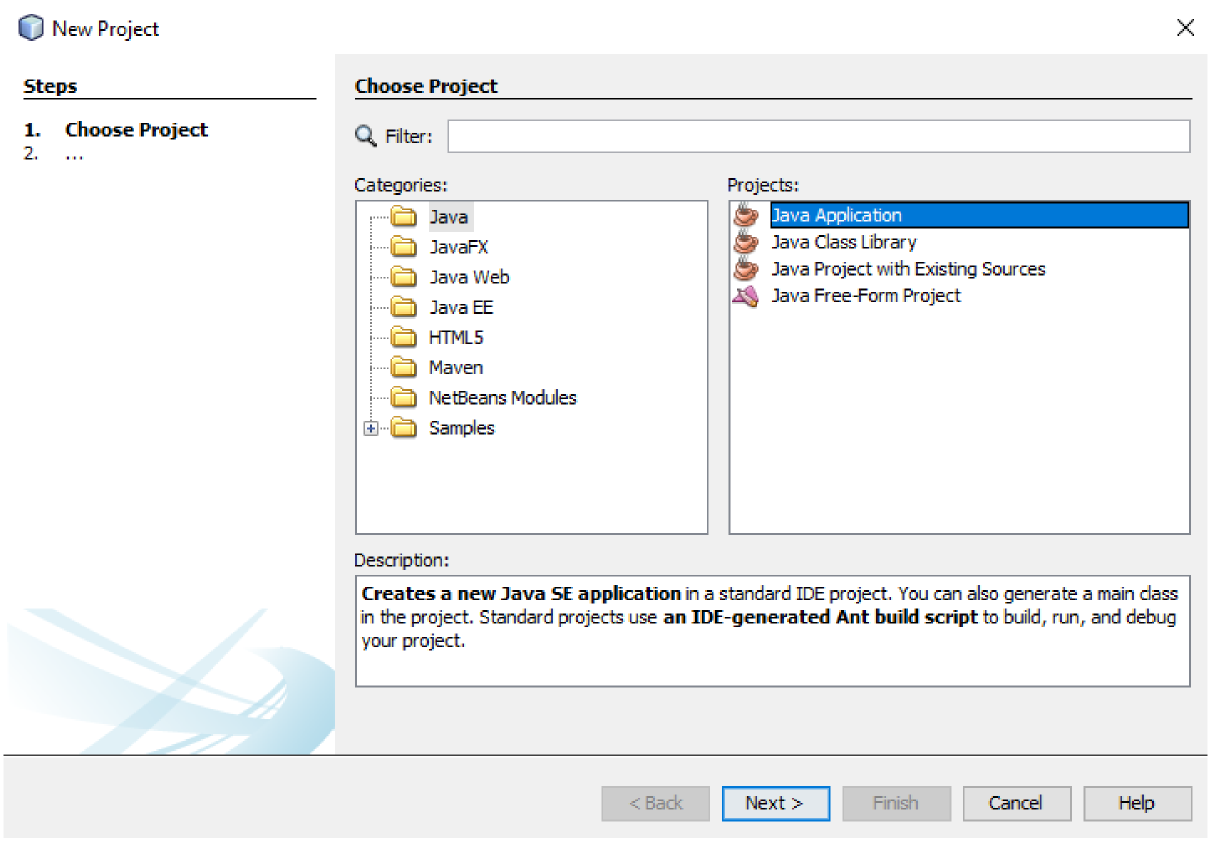
Figure 22, Testing the Web Service (Step 3)



```
Info:   Generating code...
Info:   Compiling code...
Info:   wsimport successful
Info:   Invoking wsimport with http://localhost:8080/Tutorial3/SimpleWebService?WSDL
Info:   parsing WSDL...
Info:   Generating code...
Info:   Compiling code...
Info:   wsimport successful
Info:   [SERVER] - Testing Connection...
```

Figure 23, Test Server Log

16) Now we create a client, this is going to be a separate Java Application Project (NOT A WEB APPLICATION, A SIMPLE JAVA APPLICATION) (Figure 24, Creating the Client (Step 1) and Figure 25,Creating the Client (Step 2))


New Project

Steps

1. **Choose Project**
2. ...

Choose Project

Filter:

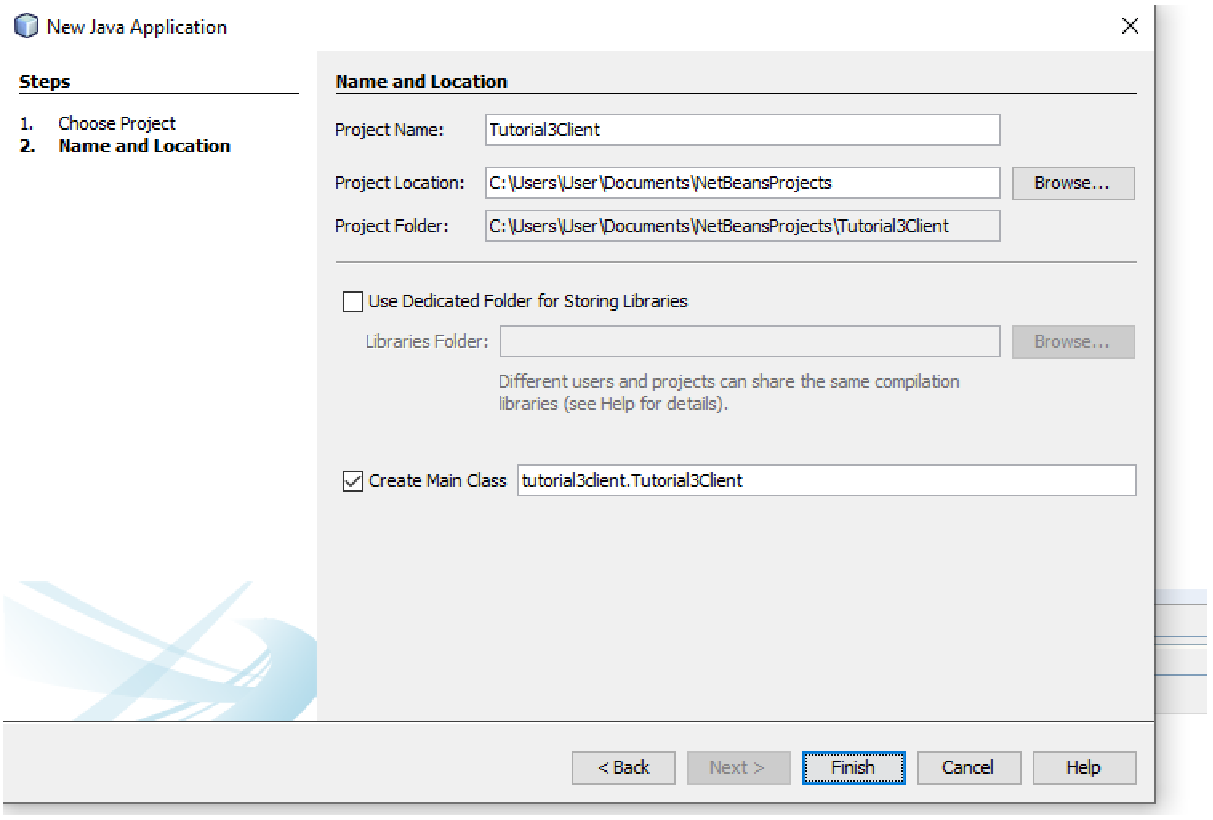
Categories: Java, JavaFX, Java Web, Java EE, HTML5, Maven, NetBeans Modules, Samples

Projects: Java Application (selected), Java Class Library, Java Project with Existing Sources, Java Free-Form Project

Description:

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 24, Creating the Client (Step 1)



Name and Location

Project Name: Tutorial3Client

Project Location: C:\Users\User\Documents\NetBeansProjects

Project Folder: C:\Users\User\Documents\NetBeansProjects\Tutorial3Client

☐ Use Dedicated Folder for Storing Libraries

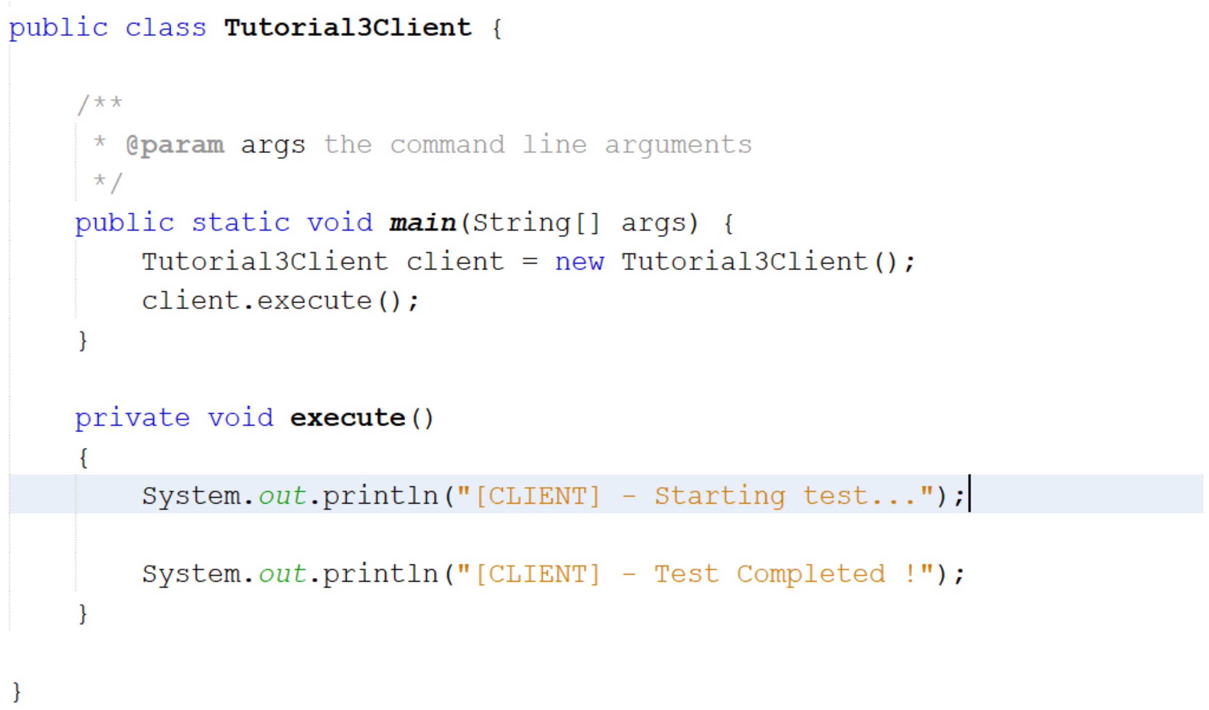
Libraries Folder:

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

☑ Create Main Class tutorial3client.Tutorial3Client

Figure 25,Creating the Client (Step 2)

17) Now we have a problem ! How can we access the server to test if it is connected (Figure 26, How can we connected to the Server ?) ? 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 (and in a separate execution space and potentially it can be deployed on a different machine). We have to build a communication system between the client and the server !



```
public class Tutorial3Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Tutorial3Client client = new Tutorial3Client();
        client.execute();
    }

    private void execute()
    {
        System.out.println("[CLIENT] - Starting test...");

        System.out.println("[CLIENT] - Test Completed !");
    }

}
```

Figure 26, How can we connected to the Server ?

18) 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, Easy Peasy !

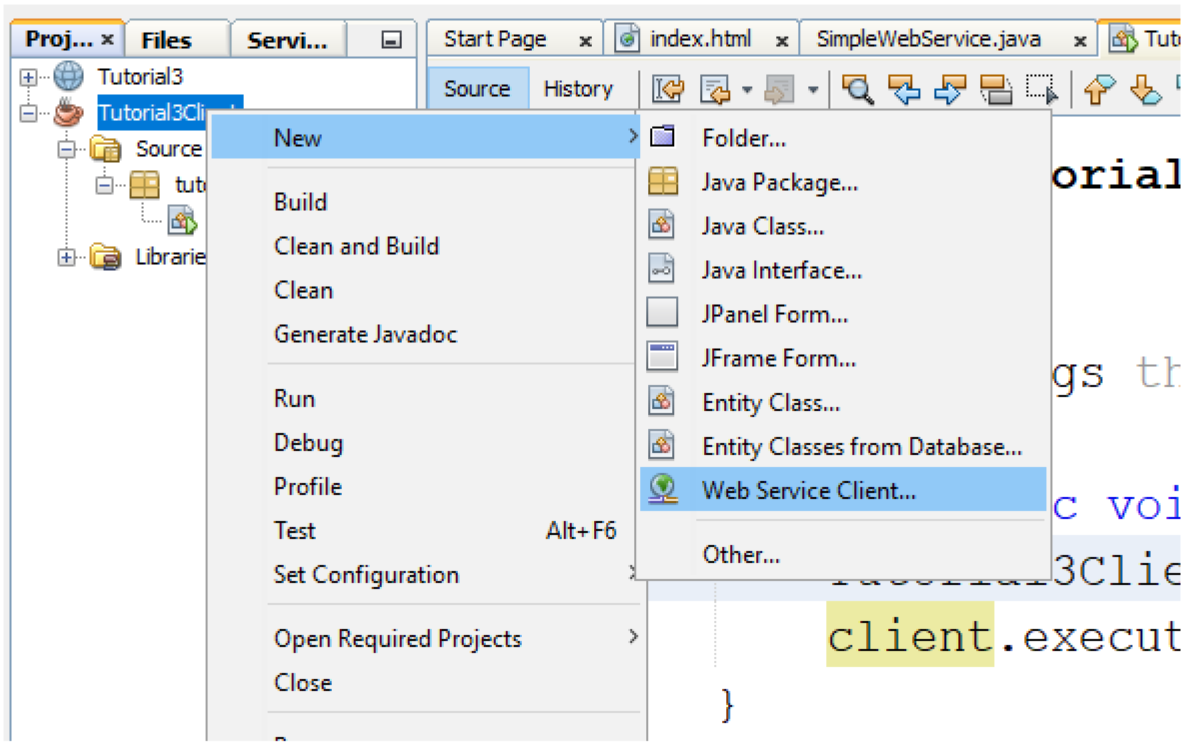
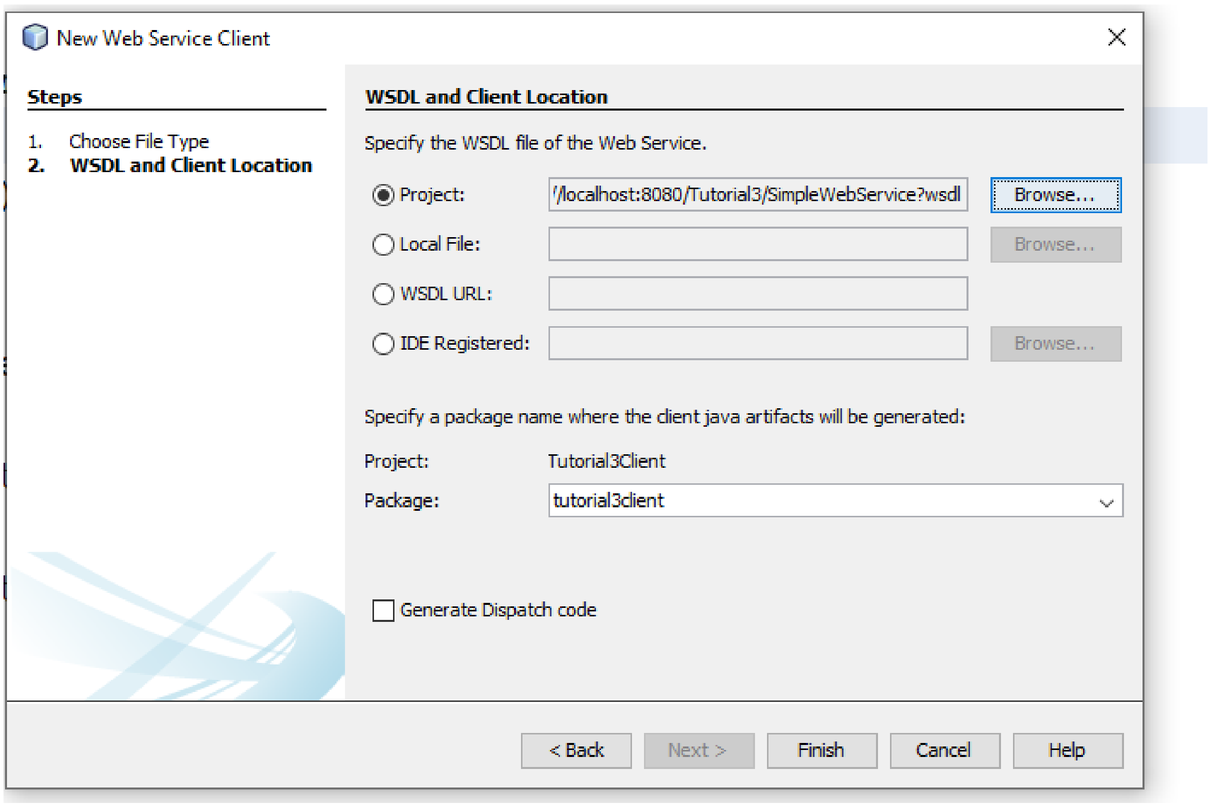


Figure 27, Creating the Web Service Client (Step 1)


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.

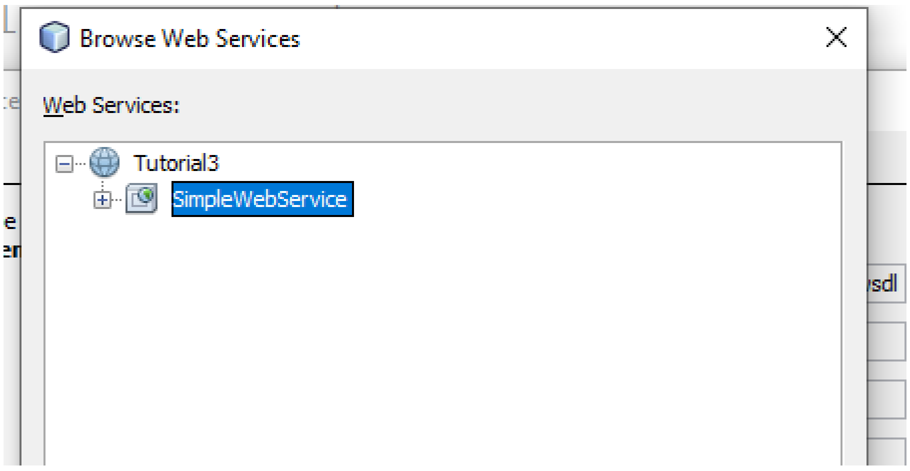
● Project: '/localhost:8080/Tutorial3/SimpleWebService?wsdl
○ Local File:
○ WSDL URL:
○ IDE Registered:

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

Project: Tutorial3Client
Package: tutorial3client

☐ Generate Dispatch code

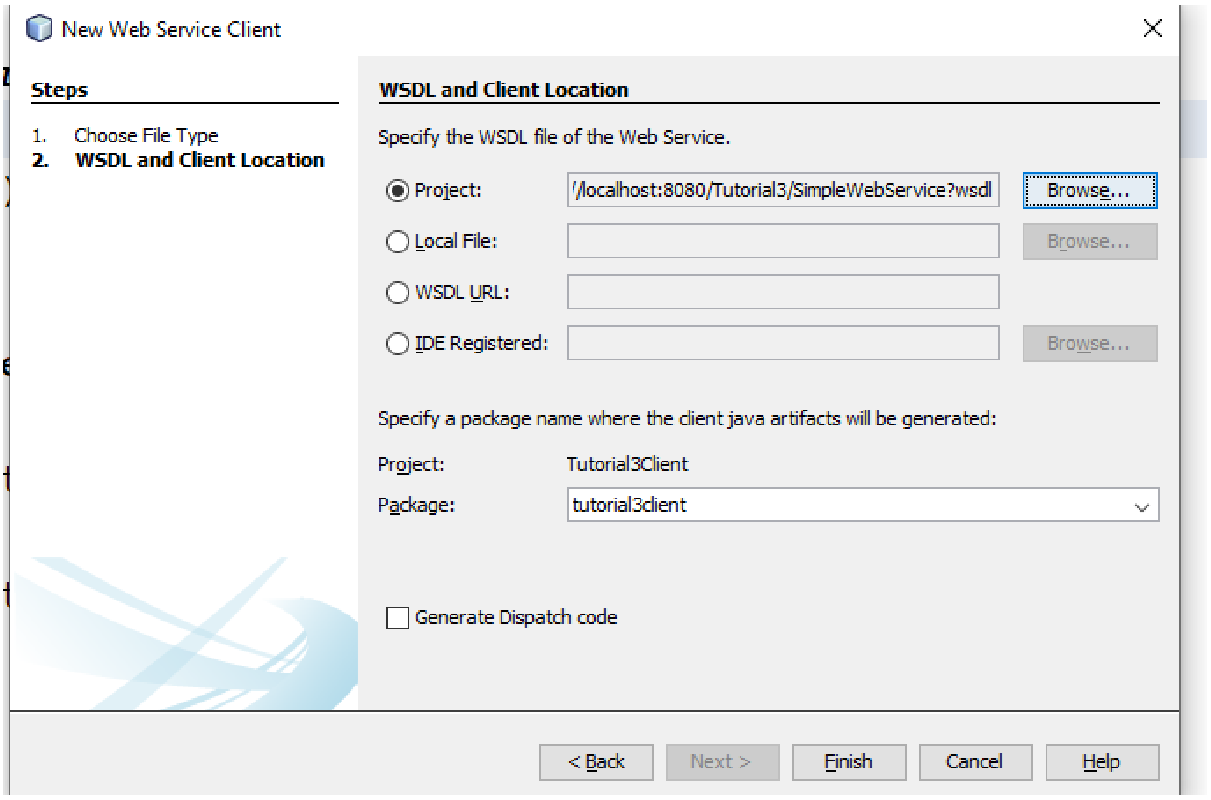
Figure 28,Creating the Web Service Client (Step 2)


Browse Web Services

Web Services:

Tutorial3
SimpleWebService

Figure 29, 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: '/localhost:8080/Tutorial3/SimpleWebService?wsdl
( ) Local File:
( ) WSDL URL:
( ) IDE Registered:

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

Project: Tutorial3Client
Package: tutorial3client

[ ] Generate Dispatch code

Figure 30, Creating the Web Service Client (Step 4)

19) 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.

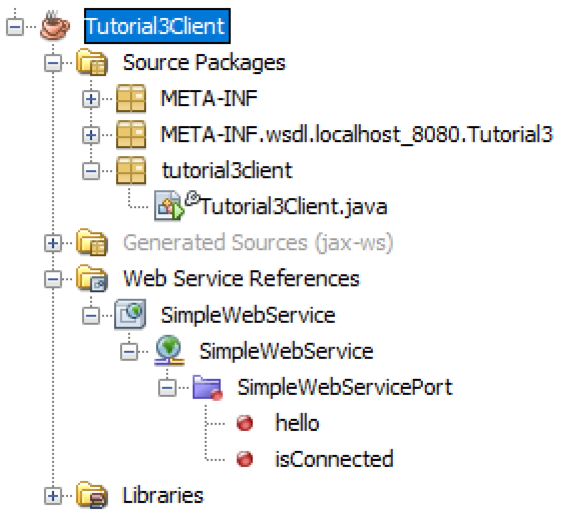
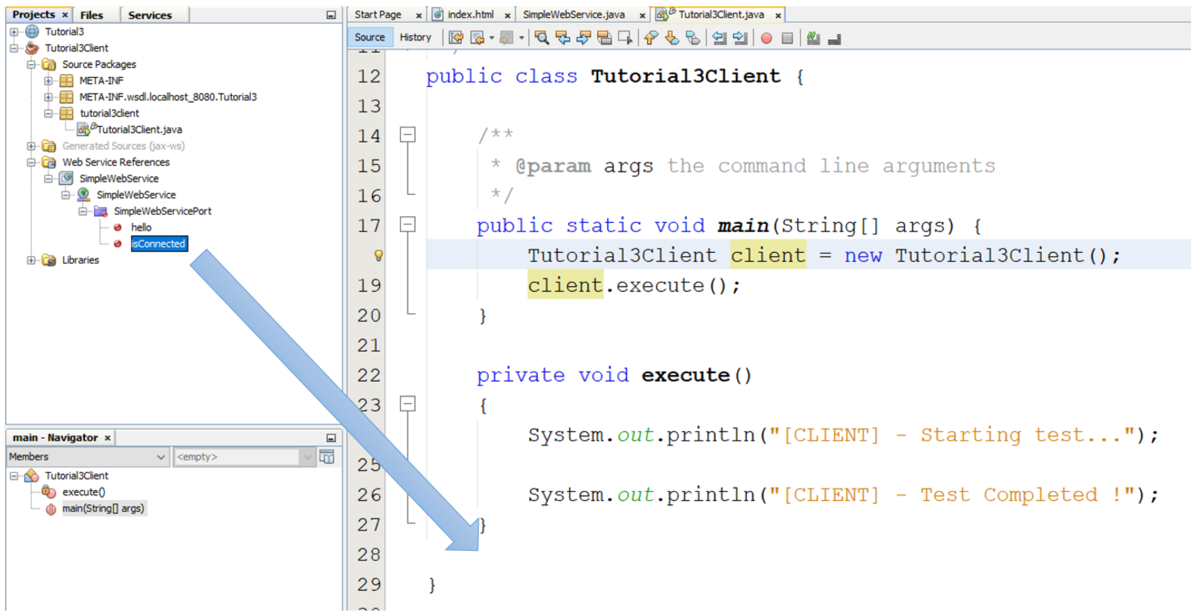

```
Tutorial3Client
  Source Packages
    META-INF
    META-INF.wsdl.localhost_8080.Tutorial3
    tutorial3client
      Tutorial3Client.java
  Generated Sources (jax-ws)
  Web Service References
    SimpleWebService
      SimpleWebService
        SimpleWebServicePort
          hello
          isConnected
  Libraries
```

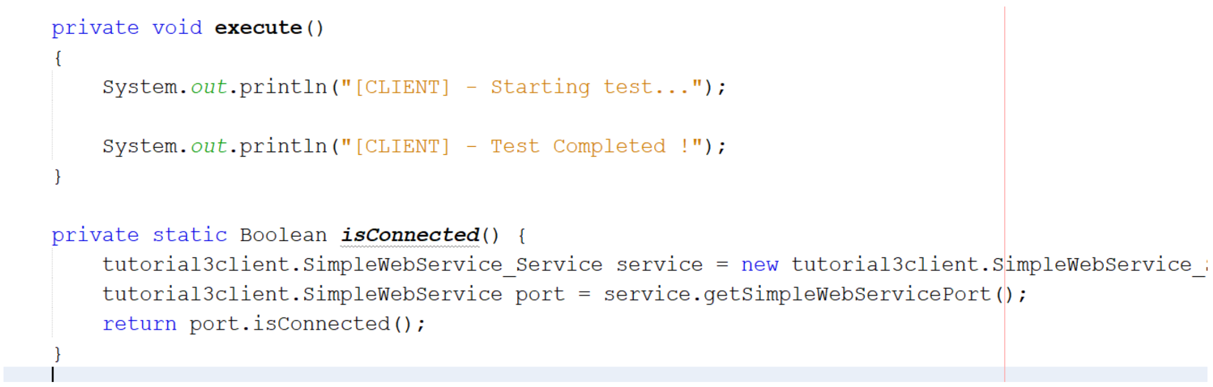
Figure 31, Client Stubs

20) 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 32, Creating Client Stubs (Step 1) and Figure 33, Creating Client Stubs (Step 2))



```
public class Tutorial3Client {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Tutorial3Client client = new Tutorial3Client();
        client.execute();
    }

    private void execute()
    {
        System.out.println("[CLIENT] - Starting test...");

        System.out.println("[CLIENT] - Test Completed !");
    }

}
```

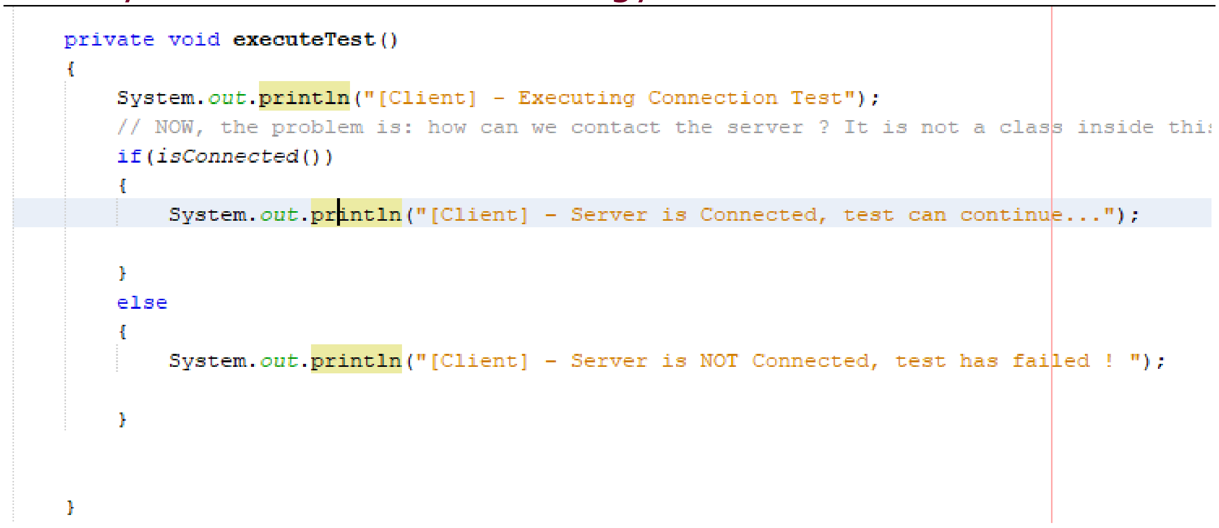
Figure 32, Creating Client Stubs (Step 1)



```
private void execute()
{
    System.out.println("[CLIENT] - Starting test...");

    System.out.println("[CLIENT] - Test Completed !");
}

private static Boolean isConnected() {
    tutorial3client.SimpleWebService_Service service = new tutorial3client.SimpleWebService_
    tutorial3client.SimpleWebService port = service.getSimpleWebServicePort();
    return port.isConnected();
}
```

Figure 33, Creating Client Stubs (Step 2)

21) Now you can invoke your client stub (which will in turn connect to the network and call the server) from your code.



```
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 34, Invoking the client stub

22) **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 into your client code (Figure 35, Copy autogenerated client stubs code. and Figure 36, Paste autogenerated client stubs code.) .

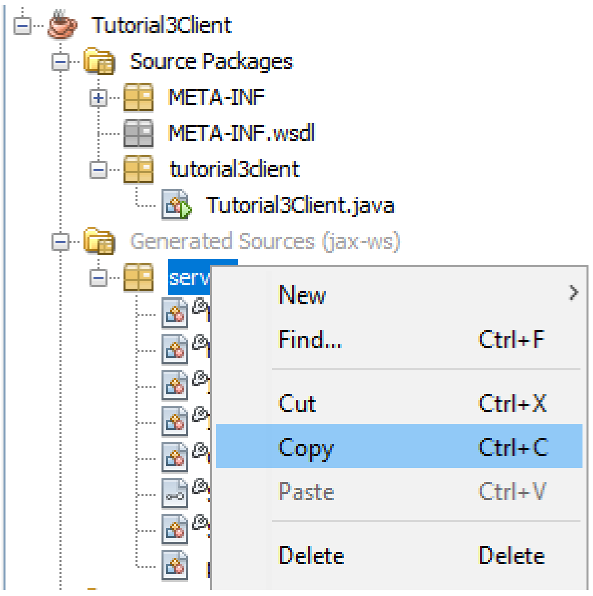


Figure 35, Copy autogenerated client stubs code.

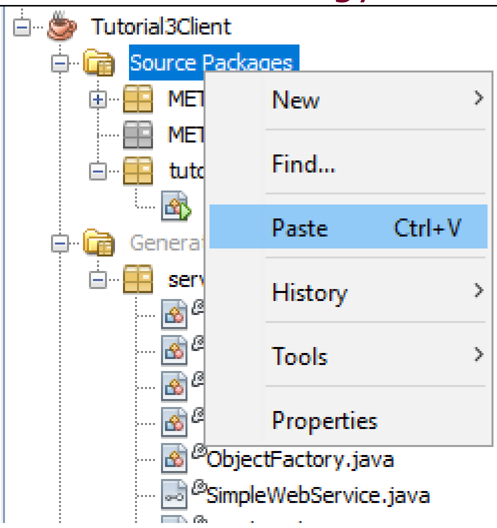


Figure 36, Paste autogenerated client stubs code.

23) 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 37, Test Client/Server Connection).

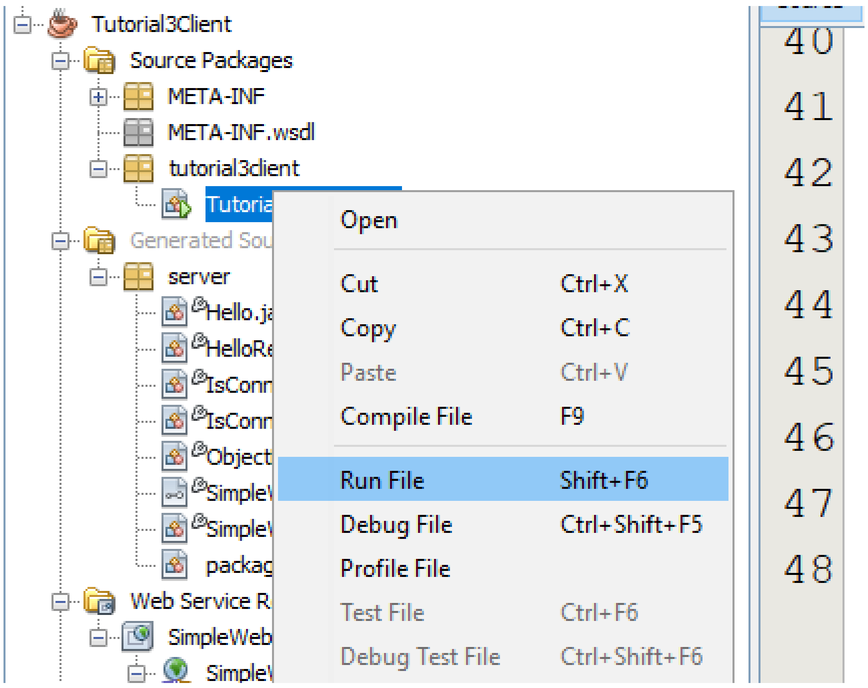
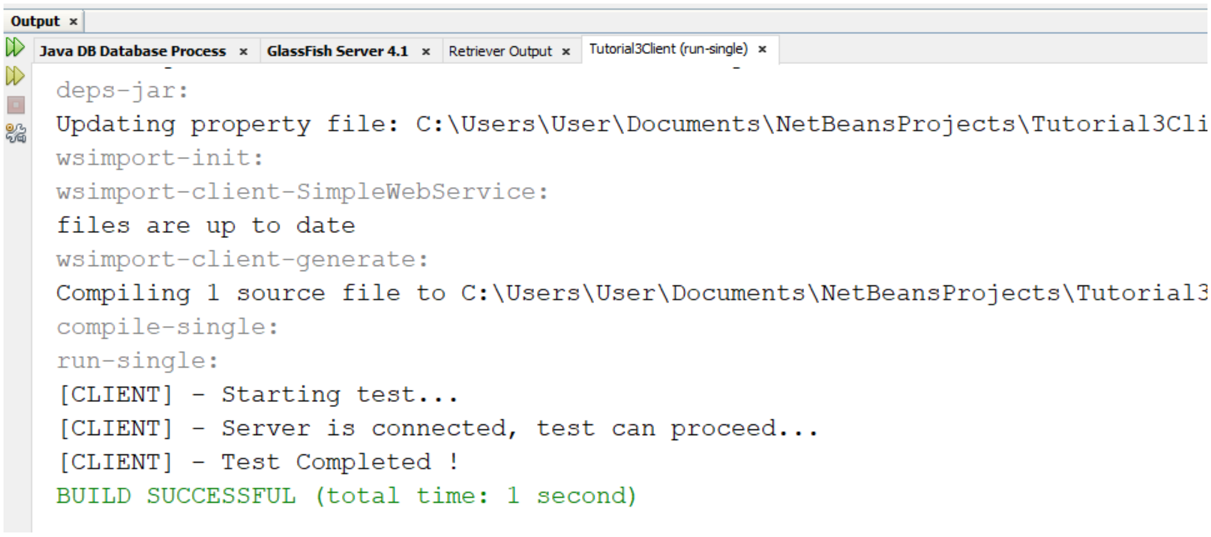


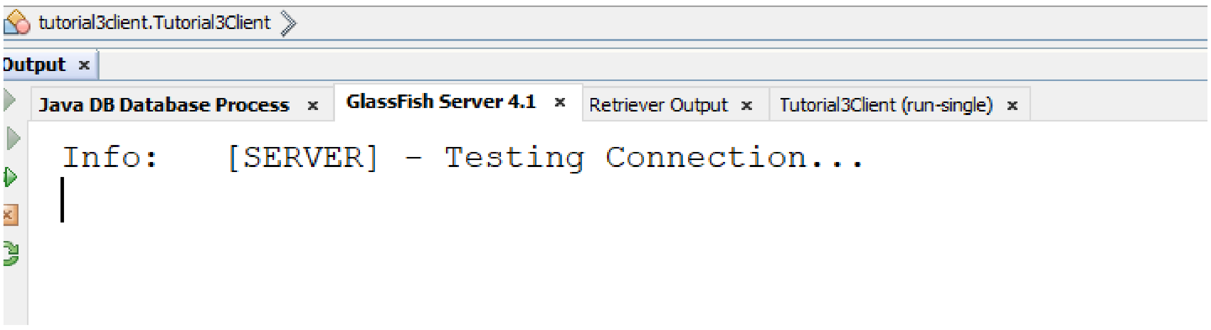
Figure 37, Test Client/Server Connection

24) 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 39, Server Log) and one from the Client (Figure 38, Client Log).



```
deps-jar:
Updating property file: C:\Users\User\Documents\NetBeansProjects\Tutorial3Cli
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 38, Client Log



```
Info:   [SERVER] - Testing Connection...
```

Figure 39, Server Log

25) 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)

26) Modify the Client so that it uses the hello method of the server to send a message from the client to the server.
27) Modify the server so that the server has a name (as a stgring) so that the server name is returned in the string from the method hello
28) 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)
29) 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
30) Investigate how you can modify the client stubs to try to connect to a server on another remote machine and try it !