Hands-On, Basic Web Service Tutorial (for Netbeans 6.8) (* 2010-06-01 draft*)

This is a basic tutorial to construct and execute a web service using a comprehensive Java-based Integrated Development Environment (IDE) called 'NetBeans'. The target audience is students with little experience with program development and programming languages. I hope this tutorial will allow MBA business students as well as IT students, to be aware of some of the underlying technologies that they will encounter or be responsible for daily in their jobs. Completing this tutorial will allow you to begin to appreciate what your developers and business analysts are describing and to contribute meaningfully to the conversation. There is no substitute for hands-on experience and this tutorial will provide that.

An instructor led development, as detailed here, will allow most students to follow the descriptions and to actually create a web service themselves, which is worth a bullet point on a resume! [2010-06-01, rob rucker]

Preliminaries to Making a Web Service that Multiplies Two Input Integers

This tutorial assumes you have the following configuration on your machine

Java 1.6 or higher, Netbeans 6.8 (netbeans 6.9 is slightly different), and a Firefox browser. (These can be downloaded for free from *java.sun.com* for the java, *netbeans.org* for the netbeans, and *mozilla.org* for the Firefox browser). Note, this tutorial will also work for Netbeans 6.5 although some of the names have changed, but the instructor can make the simple required adjustments.

For the Java download, select the SE java JDK which means Standard Edition Java Development Kit. This is now, as of May 2010, at JDK 1.6 update 20.

The netbeans IDE version is at Netbeans 6.9 RC1 (release candidate 1). For the netbeans, download the most comprehensive version available.

After downloading the java JDK, install it (just double click on the executable) taking the defaults.

After downloading the netbeans package, install it (double click on the executable) taking the defaults. Netbeans should automatically find your installed java package unless you installed it elsewhere, in which case you will have to tell netbeans where.

Note: Netbeans requires that you have the Java JDK installed first, before it will run.

Background on the Role of Integrated Development Environments (IDEs)

The key component for all the IDEs you will encounter or hear about, is a *project*. A project is a high level container that encapsulates or contains all of the materials necessary to construct a particular business application suite. So, your first activity is to create a specific project that will contain your web service as we will do below. In Netbeans, creating a project immediately gives you a detailed internal folder and file structure with most of the code and internal connections set up and filled out for you. Your task is then to build on this framework by asking for **new** components to be included within the proper section of the project. You will see this *recursive* sequence of commands starting with a command to build a project, then a web application within that, and then a particular type of web application, called a web service, within that, and then a java template file within that, and finally, a web service method within that java file. The overall plan looks like:

Overview of:

Creating a Web Service Provider

Specify Project of Category (web application)
Specify Web Service option
(template Java web service enabled class created)
Specify a web service method
(dialog box enabled that creates a template method)
The web service method template is automatically inserted within Java class
Developer writes the guts of the service method within the Java class template
Finish-
(web service code is generated (WSDL) and can now be *referenced* from a *client* over the internet)

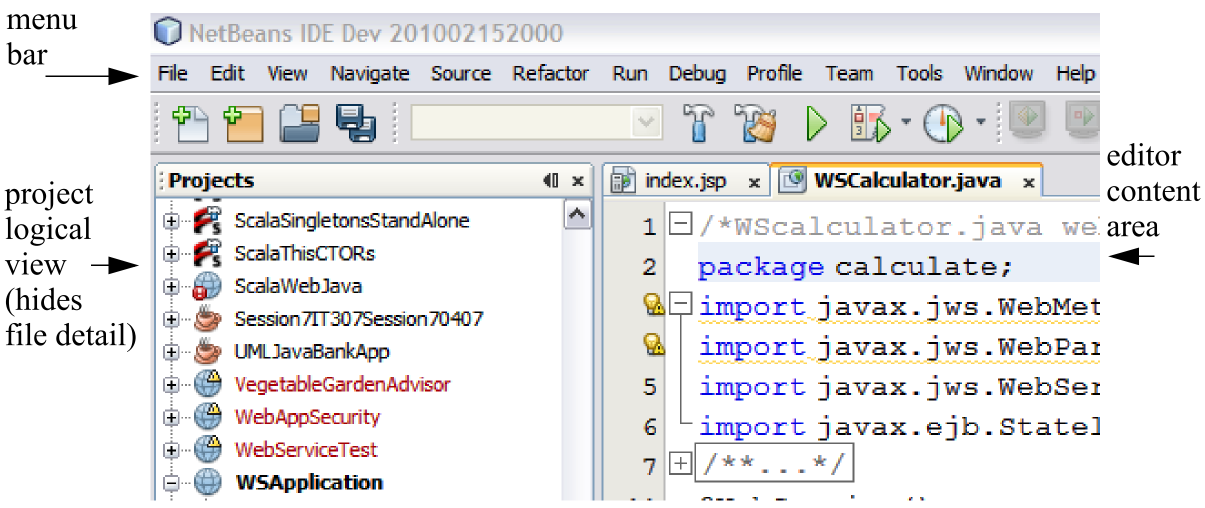
Creating a Web Service Client

Specify Project Category (java application
Specify Web Service Client option
Find WSDL of desired web service
Reference that WSDL from within a Java class
Template access code to the web service is generated.
Developer writes specific parameter values.
Client runs program, and web service return value is written out.

O.k., lets make a web service!

This project will show you how to construct an actual web service that runs over the internet. The service you will develop is very simple since, when accessed, it will only let a user client *enter two integers and return their product*! You will also see how this simple service can be improved and extended. Note that this first part of the tutorial only *creates* the service. The second phase of this tutorial creates a client that can access and *invoke (run)* this service.

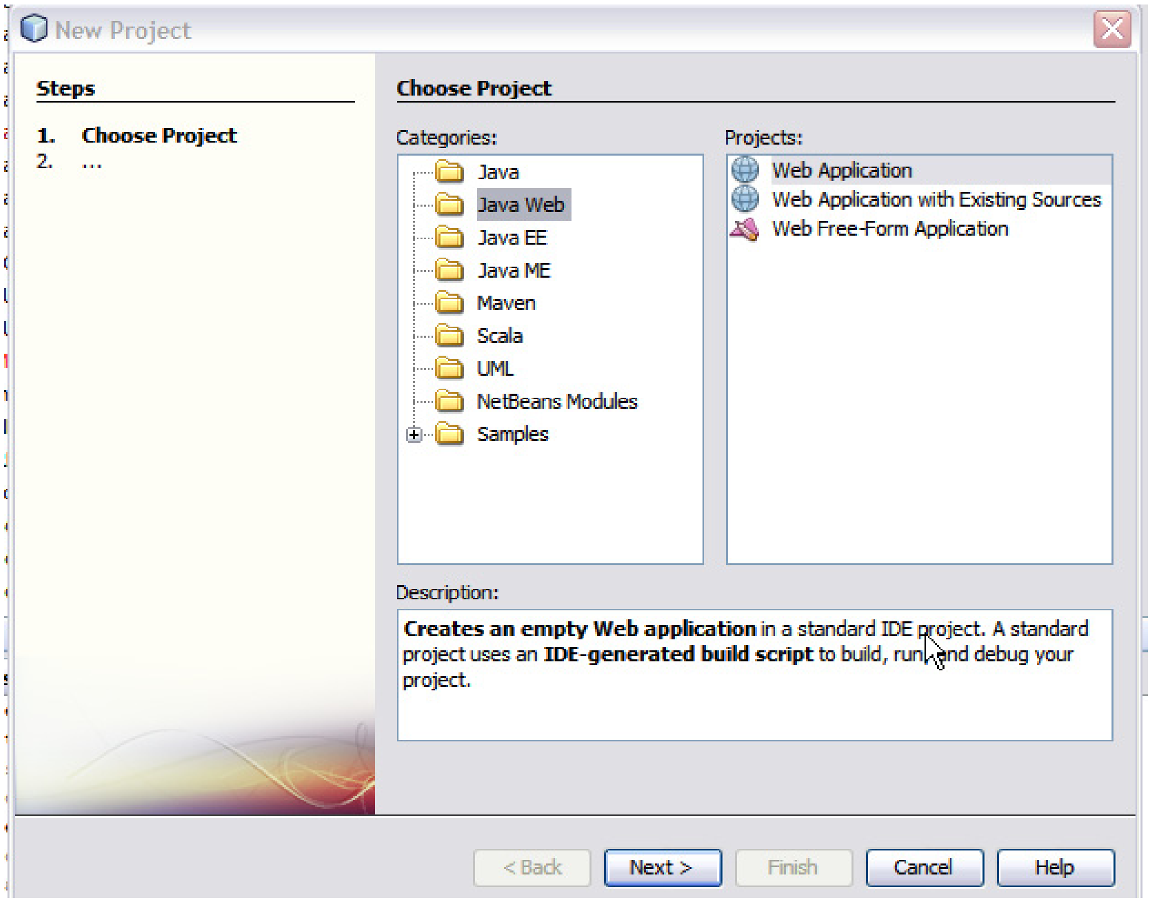
Ok, I assume netbeans is running (double click the Netbeans icon) and you can see the menu bar and the 'Projects' mode is in view. (If you don't see Projects window as shown below, go to the 'Window' menu and select 'Projects').


menu bar
project logical view (hides file detail)
editor content area
NetBeans IDE Dev 201002152000
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Projects
ScalaSingletonsStandAlone
ScalaThisCTORs
ScalaWebJava
Session7IT307Session70407
UMLJavaBankApp
VegetableGardenAdvisor
WebAppSecurity
WebServiceTest
WSApplication
index.jsp
WSCalculator.java

```
/*WScalculator.java we
package calculate;
import javax.jws.WebMet
import javax.jws.WebPar
import javax.jws.WebSer
import javax.ejb.Statel
/**...*/
```

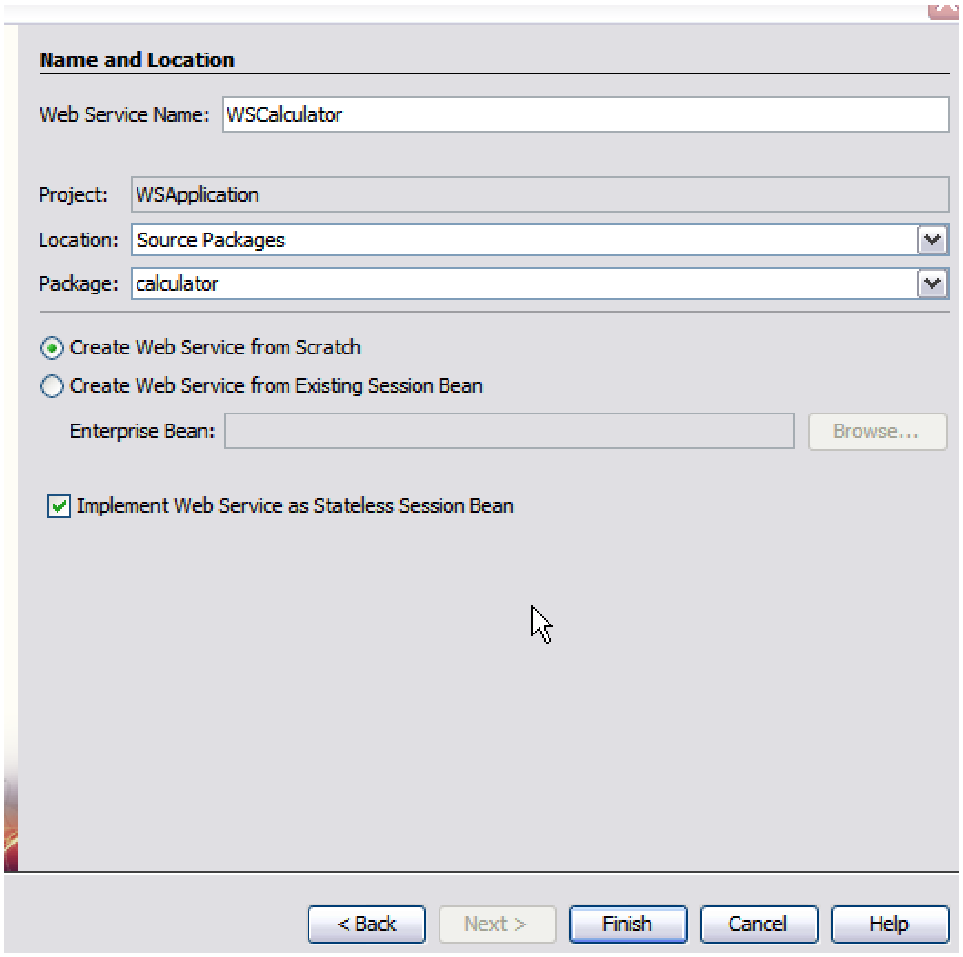
I will use the symbol '>' to indicate the next step, denoted by a 'next' button at the bottom of dialog boxes.

- Go to menu bar: File > new Project > Java Web >Web Application (this option will deploy (upload) the content of the project to an application server (Glassfish v3) that will accept requests coming in over the internet)



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.

- next > Name this project *WSApplication* > next (this next dialog panel will show *server* choices, the default is called *glassfish*) > now click finish. The result so far is a project named *WSApplication* that is set up as a *web application.*
  It will be running on a locally hosted server called *glassfish (its not running just yet)*. So far this just sets up the overall structure of a web application, we will have to further configure it to be a special kind of web application, namely, a *web service*.
- Rt-clk on the project name (*WSApplication*) > new >Categories: Web-Service > web service (this option sets up template code that you can modify for your specific web service)
- Name the web service *WSCalculator* and type in a package name of *calculator*. The dialog box looks like:


Name and Location

Web Service Name: WSCalculator

Project: WSApplication

Location: Source Packages

Package: calculator

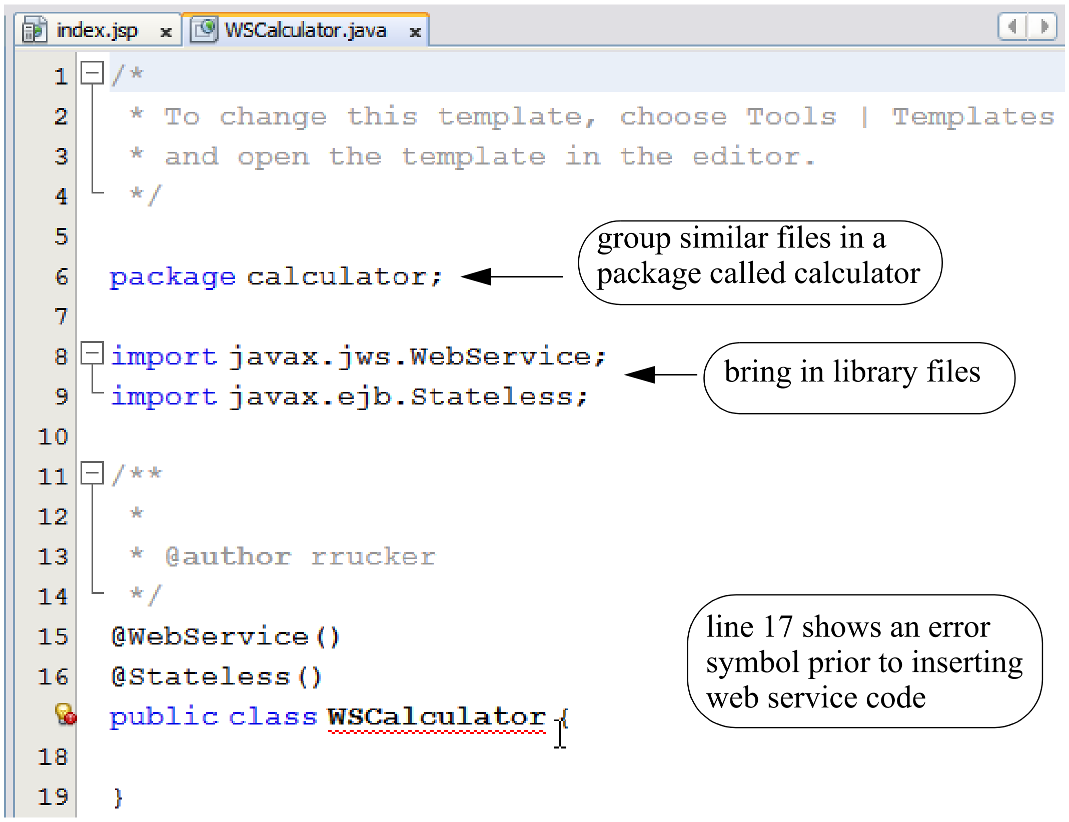
[● Create Web Service from Scratch]

○ Create Web Service from Existing Session Bean

Enterprise Bean:

[✓ Implement Web Service as Stateless Session Bean]

- *check* the radio button that says create web service from scratch and *check* the radio button that says Implement Web Services as a stateless Session Bean.
- click finish. At this point, the IDE creates a template Java file named *WSCalculator.java*, that is configured to allow construction of a web service within it. You will need to supply the details of the web service operations, but the template is created for you and is shown below.


index.jsp
WSCalculator.java
group similar files in a package called calculator
bring in library files
line 17 shows an error symbol prior to inserting web service code

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package calculator;

import javax.jws.WebService;
import javax.ejb.Stateless;

/**
 *
 * @author rrucker
 */
@WebService()
@Stateless()
public class WSCalculator {

}
```

- Look at the Java File above and place your cursor between the braces '{'and '}'. Rt-clk and select insert code > add business method.

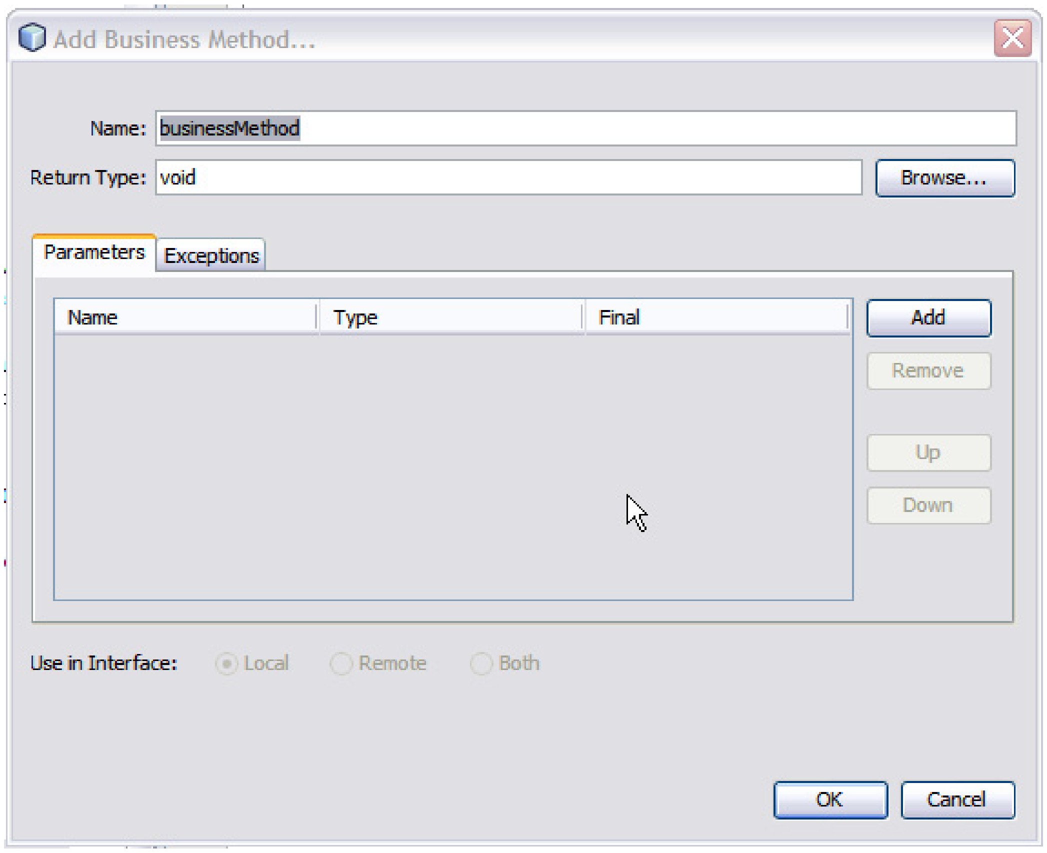
  Keep in mind that a 'method' has a name, a specification of parameters for input, manipulation of those parameters within the body of the method, and a return of some kind of result. The dialog box shown below allows a name to be specified, the return type, and the names and types of the input parameters. Notice that there is no specification of what to *do* with these parameters, that is done in the Java code which we will get to after this template is filled out.

- A dialog box comes up allowing you to specify a template for the web service method. Type in a name for this web service, say, *multiply*. Type in an *int* for the return type and type in two lines of parameters to use as multipliers. Here are the detailed steps:

1. You supply the name of the web service, in our case type in *multiply*.
2. What do I want my method to return to the client? Since it is a multiplication of two integers, I want to return their product, and hence the return type ought to be an *integer (int)*.
3. Now press the Add button to specify the name and type of an input parameter. The first such parameter is an integer (int) that I named 'a'. Pressing Add again allows me to specify another input integer I named 'b'. Therefore, the method *multiply*, is prepared to receive two integers, which I called 'a' and 'b'. (Not terribly creative, you can do better!).

Web Service Creation and Use

•


Add Business Method...

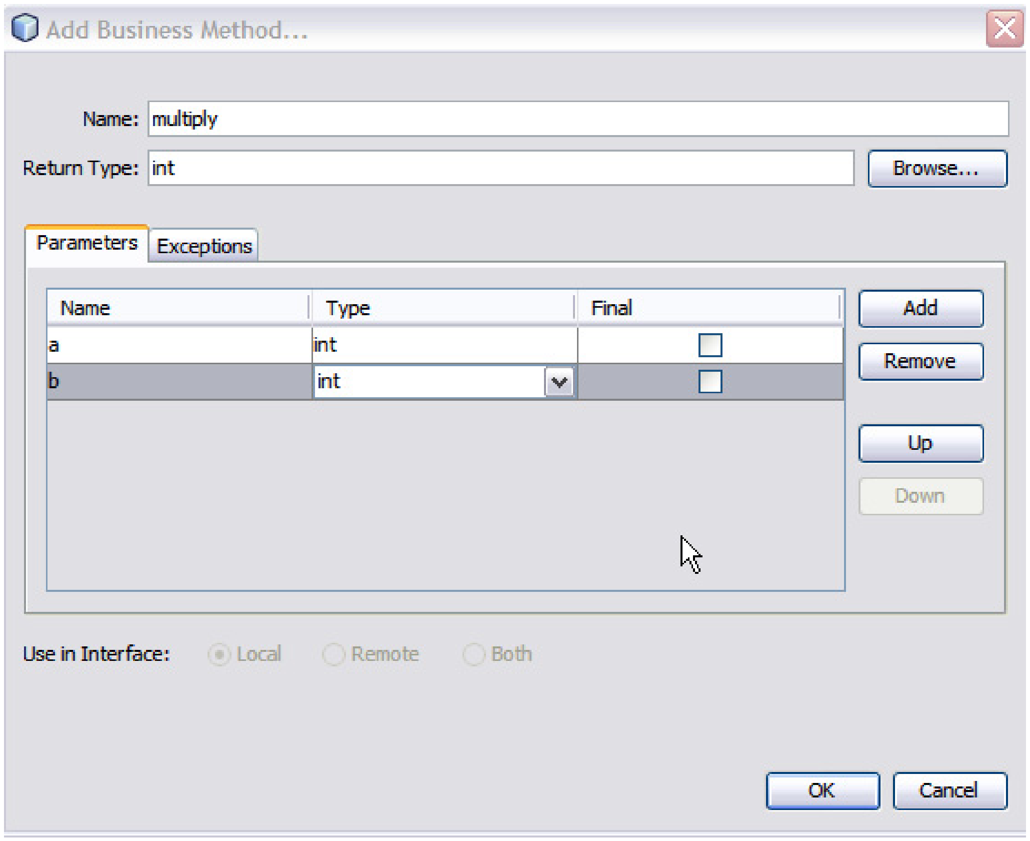
Name: businessMethod

Return Type: void

Name	Type	Final

Use in Interface: [✓ Local] Remote Both

- Fill in this dialog box as below:



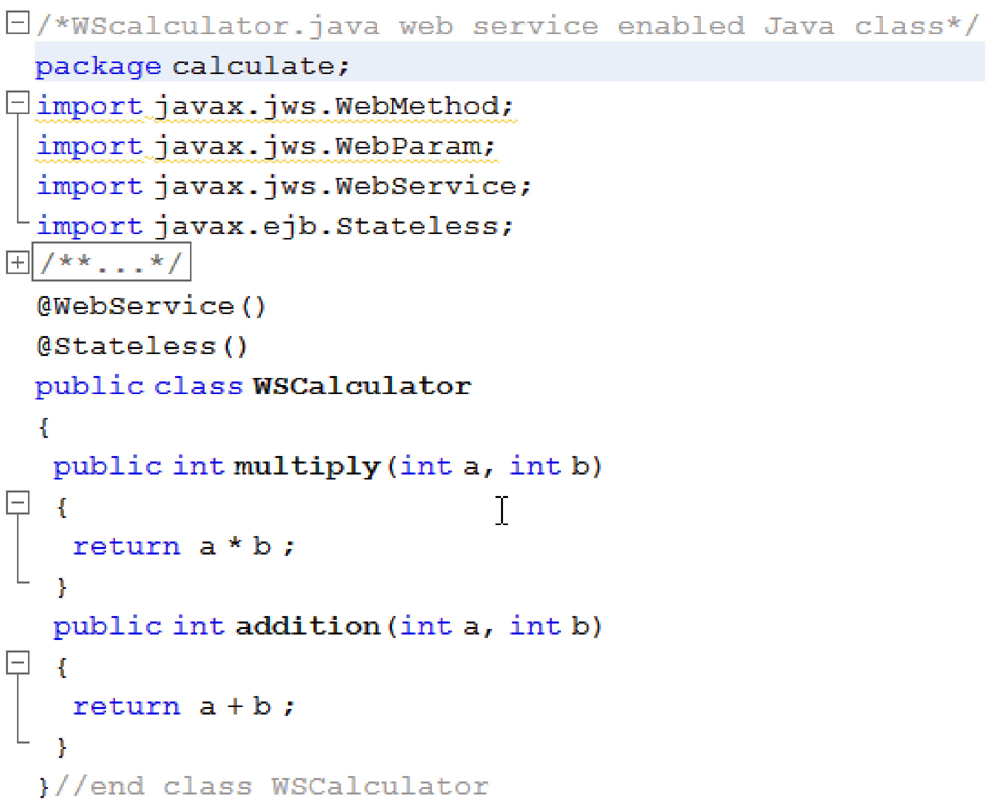
Name: multiply

Return Type: int

Name	Type	Final
a	int	☐
b	int	☐

Use in Interface: [✓ Local] Remote Both

- click OK and the resultant Java file looks like the one below (I have added in a second web service called addition just to show how you can extend this file).



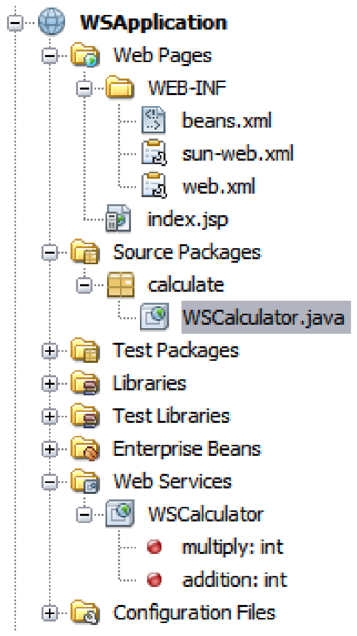
```
/*WScalculator.java web service enabled Java class*/
package calculate;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.ejb.Stateless;
/**...*/
@WebService()
@Stateless()
public class WSCalculator
{
 public int multiply(int a, int b)
 {
  return a * b ;
 }
 public int addition(int a, int b)
 {
  return a + b ;
 }
}//end class WSCalculator
```

Testing Your Web Service

To test out your *multiply* web service method, do the following: (remember you are constructing a web service which means you must start it running in a server that is connected to the internet, or at least to your internal network).

- Rt-clk on the project name and select *deploy.*
- This command will start up the glassfish server and instantiate your web service inside the glassfish server, ready to be accessed over the web. The server is running on your machine at an address of *localhost. where local host resolves to:* Internet address == 127.0.0.1
- Now go into the project view and navigate to the folder labeled W*eb Services*. Within that you will see *WSCalculator* now represented as its webservice role and below that a red dot denoting the multiply method. (I also created an addition web service which shows up as a red dot also). The project should now look as below. Notice that there was a new folder auto-generated called *Web Services*:

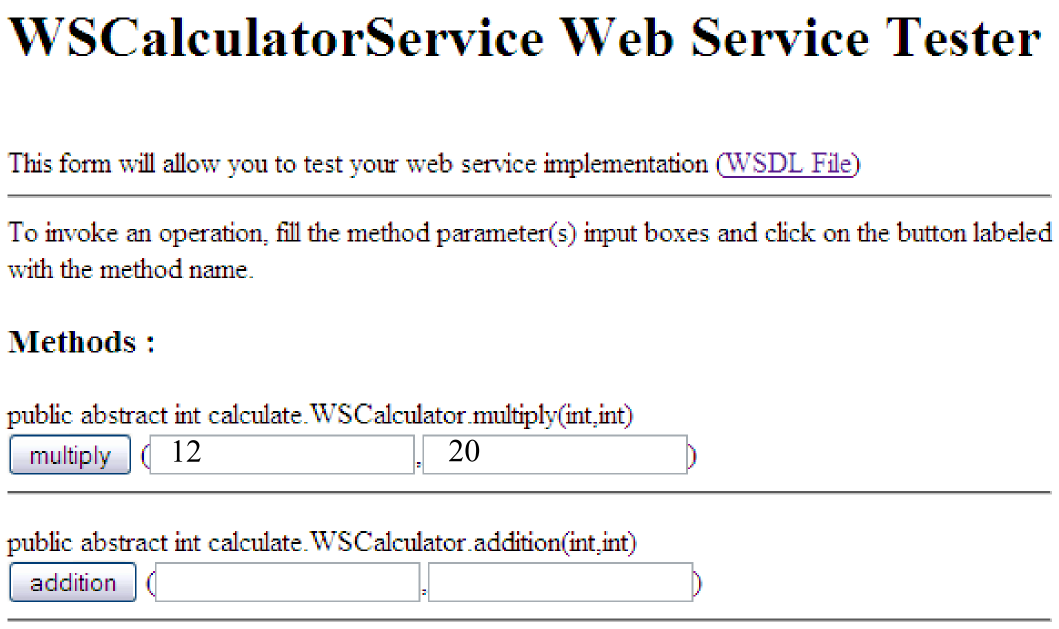
•



- Rt-clk on WSCalculator (under the *Web Services* folder) and select Test Web Service

A Firefox browser window will open asking you to input two integers to test your calculate method. Try this out with various numbers to see it working. (What happens if you enter some alphabetic characters?). Note: if you wish to use another browser such as Chrome, or Internet Explorer, then go to the tools>options>General>default browser> select desired browser.

menu and select default browser option and select your choice.


WSCalculatorService 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.

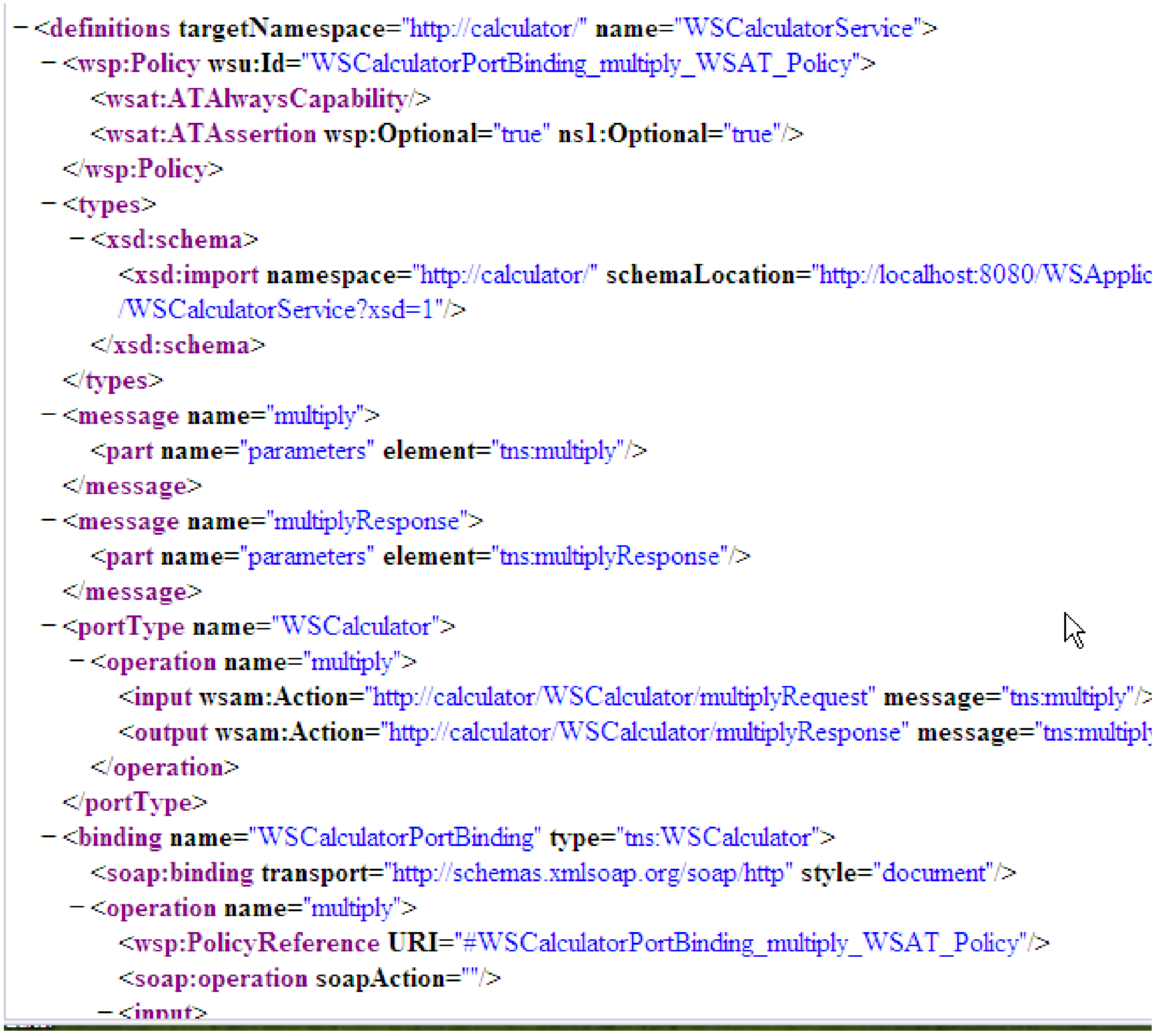
Methods :

public abstract int calculate.WSCalculator.multiply(int,int)

public abstract int calculate.WSCalculator.addition(int,int)

WSDL File (a Partial View)

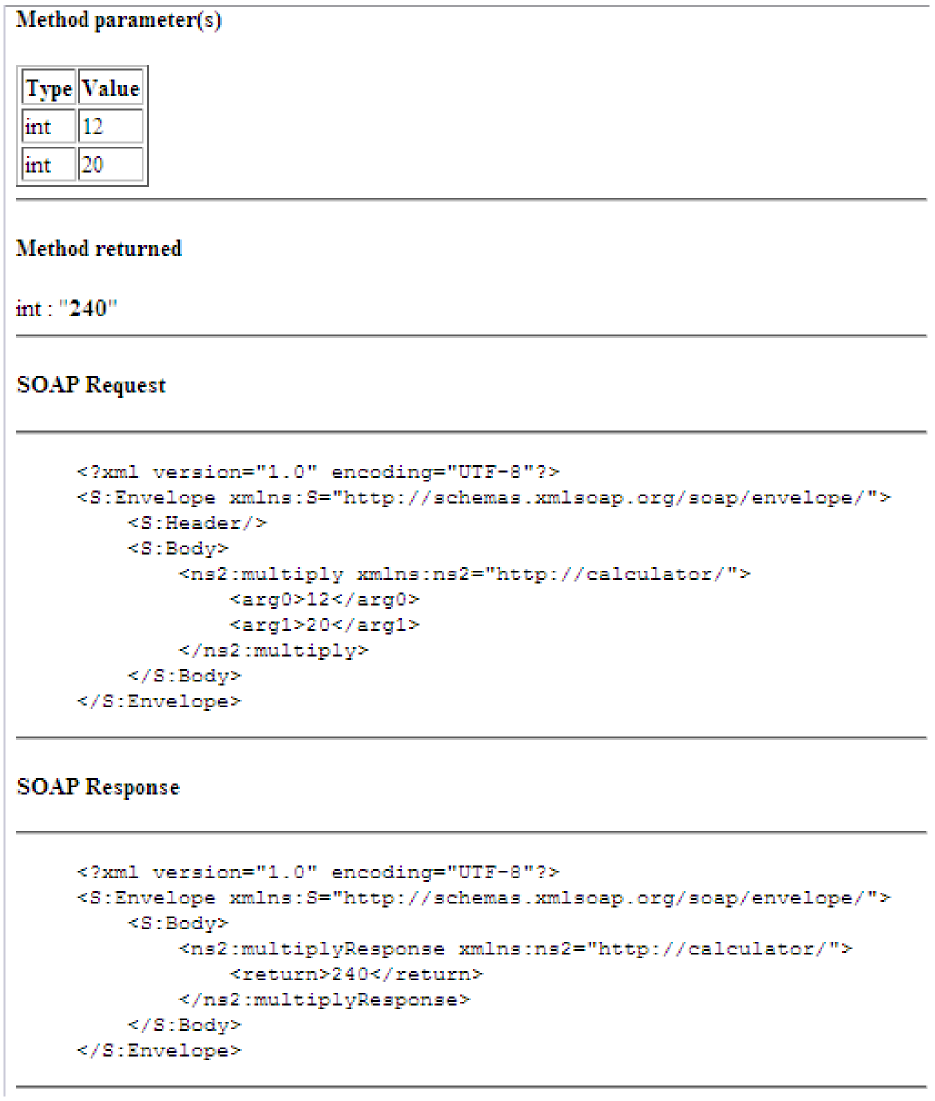
As a part of a web service, remember that there is a WSDL Web Services Definition Language) file that details the properties of the service so that a remote client can access this WSDL file, download it, and then use the WSDL instructions in order to determine how to access the service. In this case the service consists of multiplying two supplied integers. Below is the WSDL file for the two web services I developed.



```
- <definitions targetNamespace="http://calculator/" name="WSCalculatorService">
  - <wsp:Policy wsu:Id="WSCalculatorPortBinding_multiply_WSAT_Policy">
      <wsat:ATAlwaysCapability/>
      <wsat:ATAssertion wsp:Optional="true" ns1:Optional="true"/>
    </wsp:Policy>
  - <types>
    - <xsd:schema>
        <xsd:import namespace="http://calculator/" schemaLocation="http://localhost:8080/WSApplic
        /WSCalculatorService?xsd=1"/>
      </xsd:schema>
    </types>
  - <message name="multiply">
      <part name="parameters" element="tns:multiply"/>
    </message>
  - <message name="multiplyResponse">
      <part name="parameters" element="tns:multiplyResponse"/>
    </message>
  - <portType name="WSCalculator">
    - <operation name="multiply">
        <input wsam:Action="http://calculator/WSCalculator/multiplyRequest" message="tns:multiply"/>
        <output wsam:Action="http://calculator/WSCalculator/multiplyResponse" message="tns:multipl
      </operation>
    </portType>
  - <binding name="WSCalculatorPortBinding" type="tns:WSCalculator">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    - <operation name="multiply">
        <wsp:PolicyReference URI="#WSCalculatorPortBinding_multiply_WSAT_Policy"/>
        <soap:operation soapAction=""/>
      - <input>
```

SOAP

To effect this web service, messages are sent from the application to the web server and back. The protocol for these messages is SOAP.


Method parameter(s)

Type	Value
int	12
int	20

Method returned

int : "**240**"

SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:multiply xmlns:ns2="http://calculator/">
            <arg0>12</arg0>
            <arg1>20</arg1>
        </ns2:multiply>
    </S:Body>
</S:Envelope>
```

SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:multiplyResponse xmlns:ns2="http://calculator/">
            <return>240</return>
        </ns2:multiplyResponse>
    </S:Body>
</S:Envelope>
```

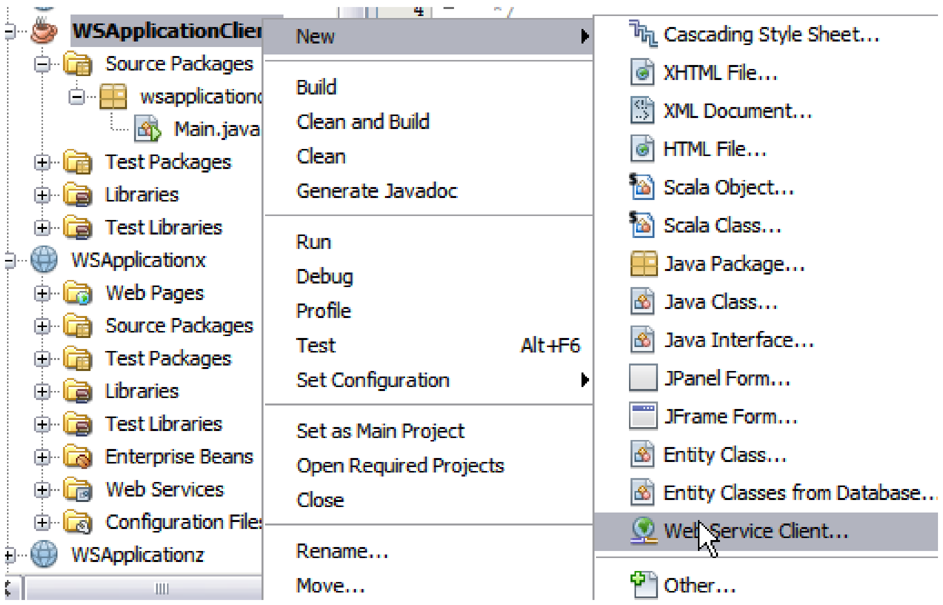
Phase II -- Accessing the Web Service as a Client

Now we have our web service, *multiply*, but how can somebody else access and use it? That is the topic of this next section.

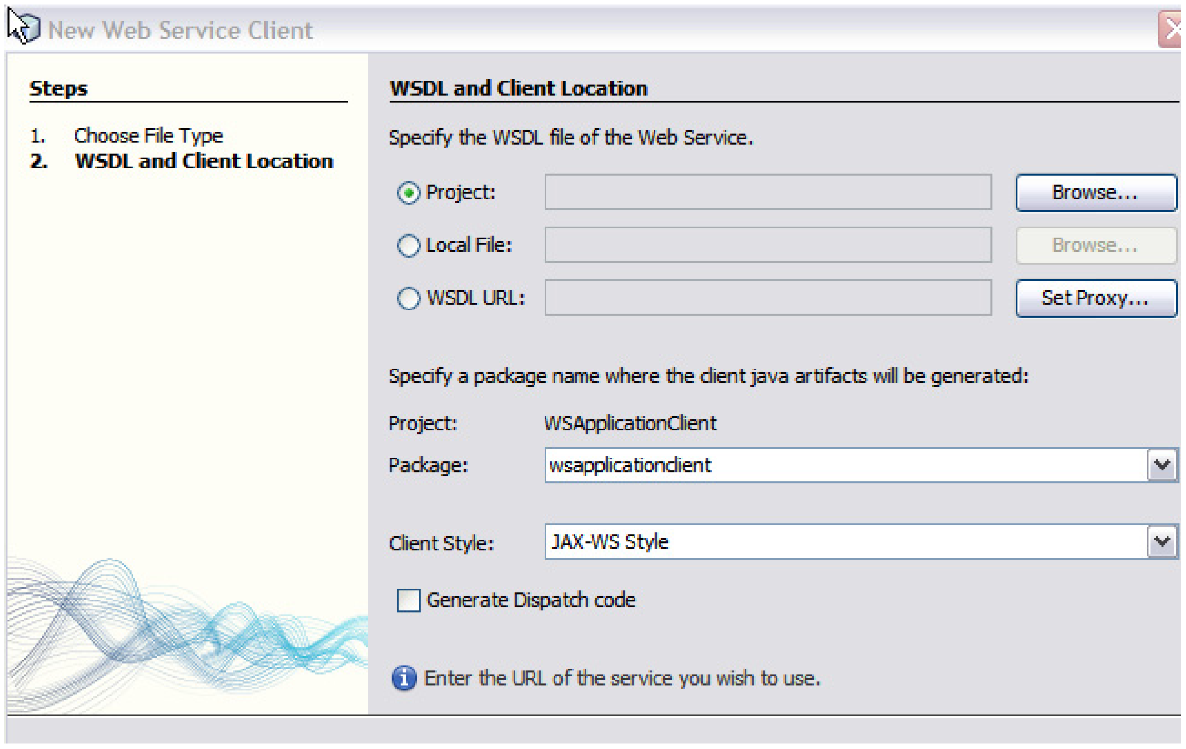
Constructing the Client Project

Before you start the client project, go back to your WSApplication project Rt-clk on the project name and select ‘deploy’. This will start the glassfish server going so that the client will have access to that web applications’ components.

- File > new Project > Categories: Java > ‘select the Java Application (click on it)’ >next ‘name the project’ *WSApplicationClient* > ‘accept all the defaults’ >finish
- Rt-clk on project node (WSApplicationClient) >New > Web Service Client



At this point you will see a dialog box as below.


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: [ ] Set Proxy...

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

Project: WSApplicationClient

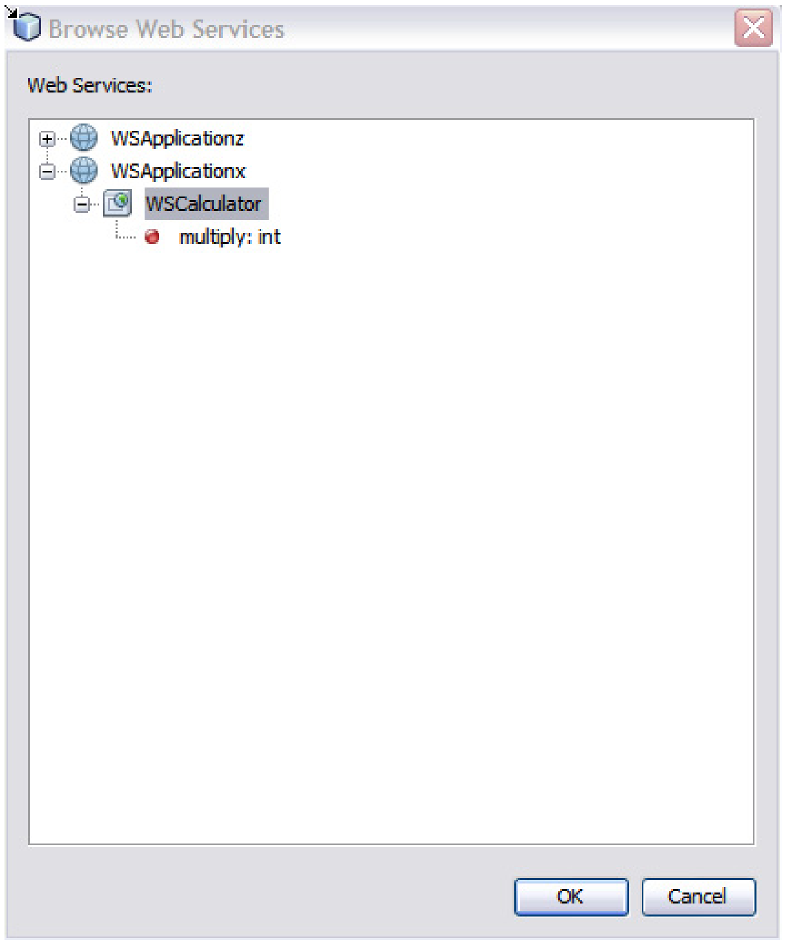
Package: wsapplicationclient

Client Style: JAX-WS Style

[ ] Generate Dispatch code

Enter the URL of the service you wish to use.

- Make sure you select the package you specified in your client, (it will be in the drop down list)
- When you click browse, a dialog box will open with projects containing web services. Select your project (*WSApplication*), open it up and select *WSCalculator.*



- Click ‘ok’ and, back in the previous dialog box, the address of the needed URL to the WSDL file will be filled in as below:

```
http://localhost:8080/WSApplicationx/WSCalculatorService?wsdl
```

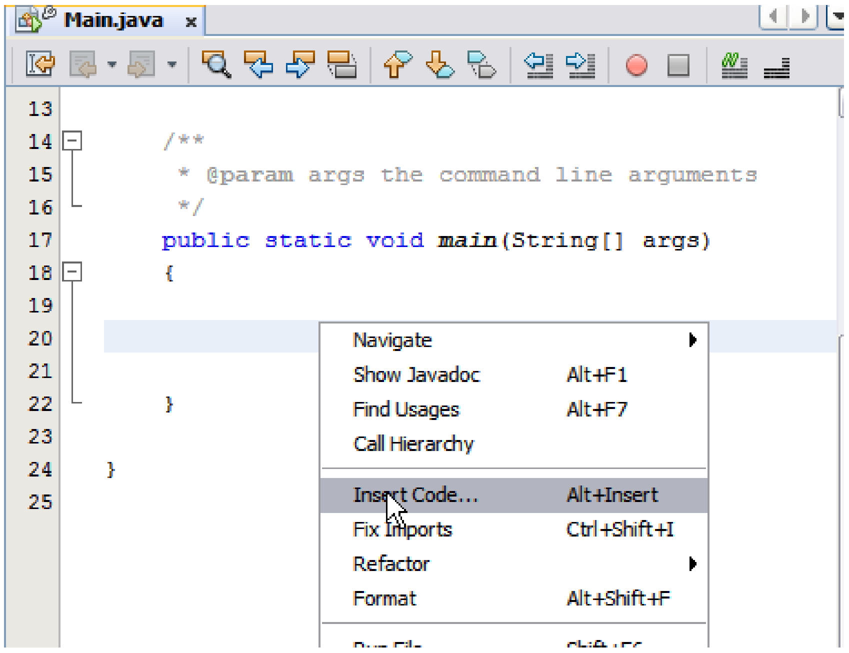
- Click finish

In your client class, you now have a reference to your WSApplication Web Service

At this point, you have created a *Web Services Reference folder* that contains a reference to the WS-Calculator web service (recall that this web service was created earlier by you within the *WSAppli-cation* project.

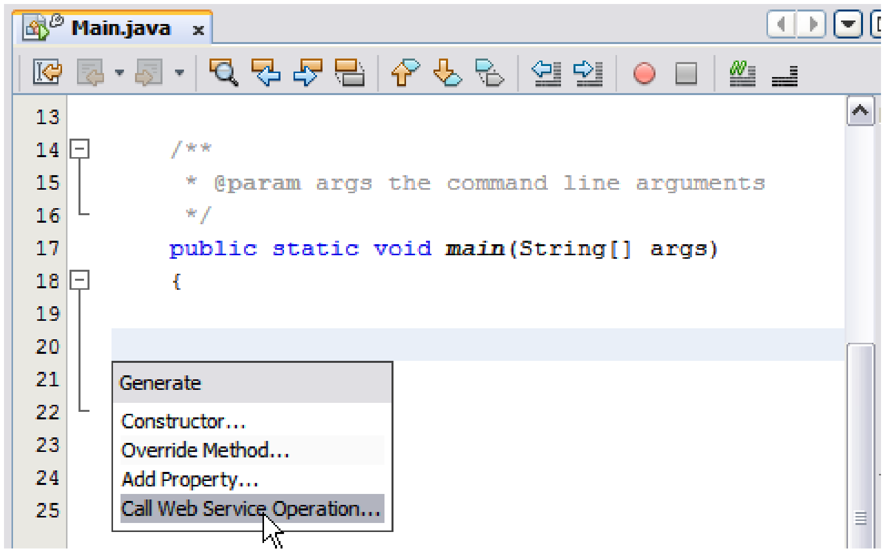
- Rt-clk the Main.java class so that it opens in the editor. You will see the choices as below:

- Place your cursor between the braces '{', '}'

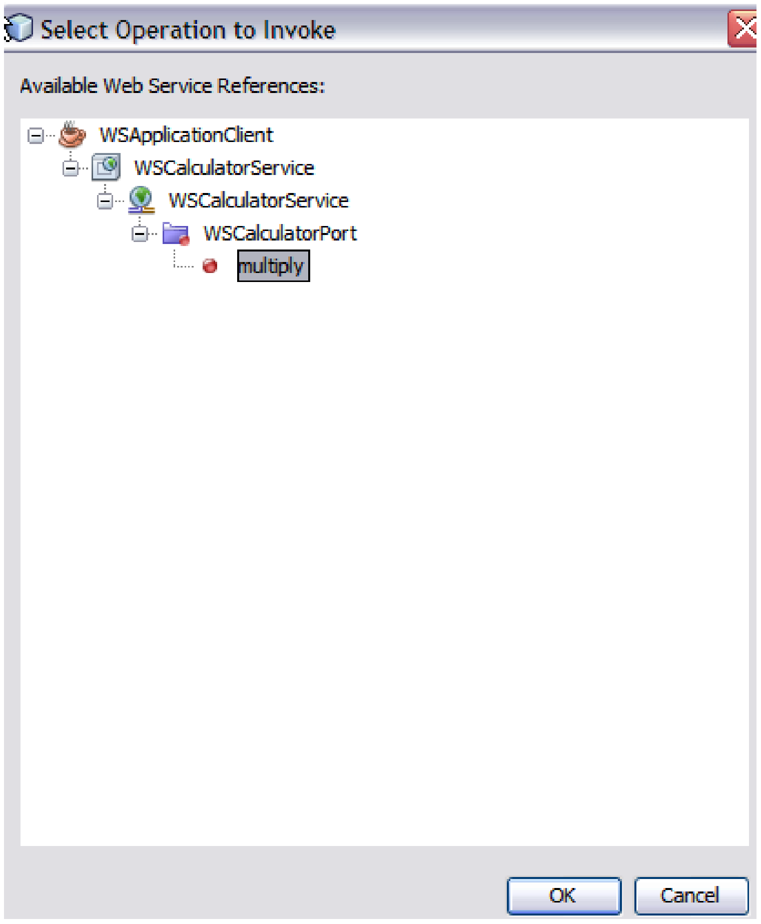


```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {


    }

}
```

Clicking on *insert code* yields:



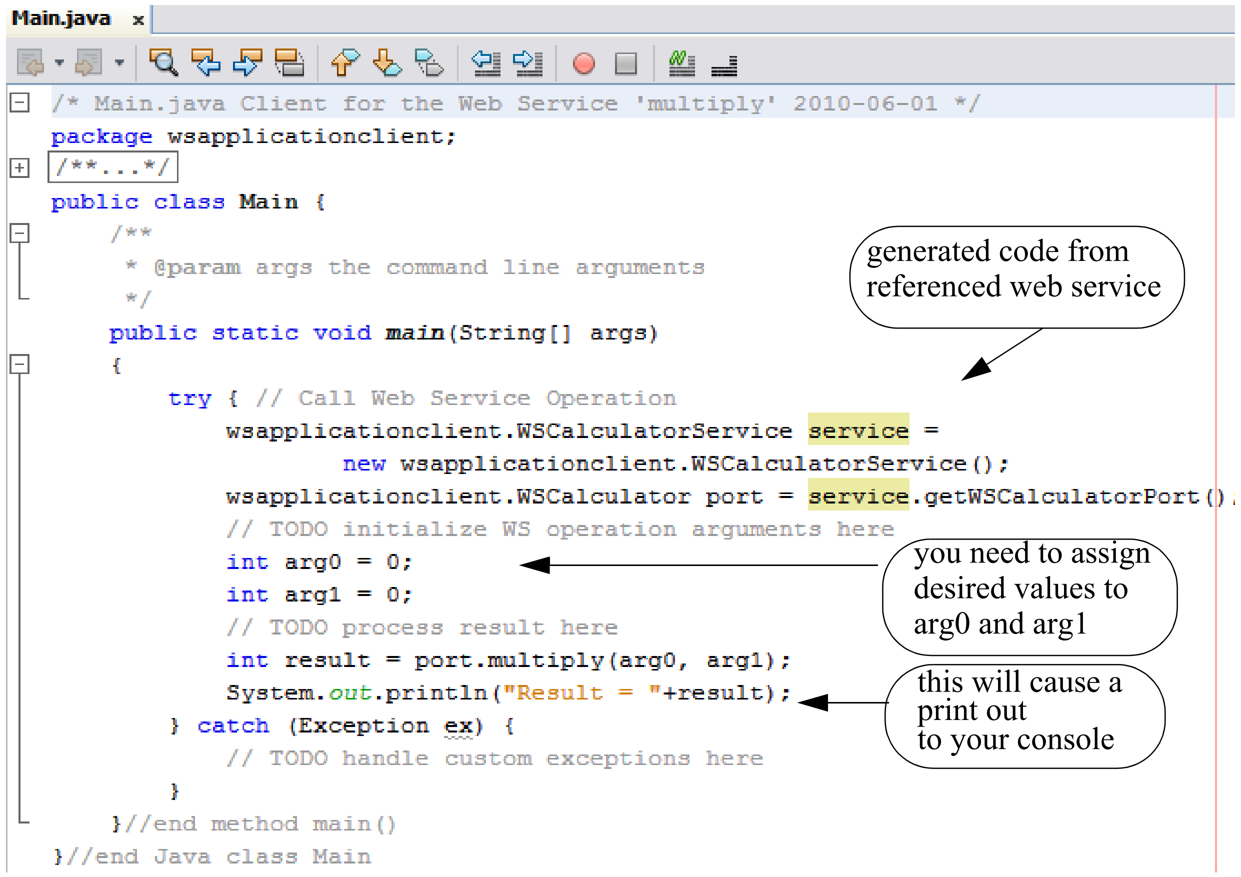
```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
```



Select your web service and Clk Ok

After this 'OK, your Main method will be filled in with the code shown below:

At this point, 'arg0' and 'arg1' are the translated versions of our 'a' and 'b' parameters established earlier.



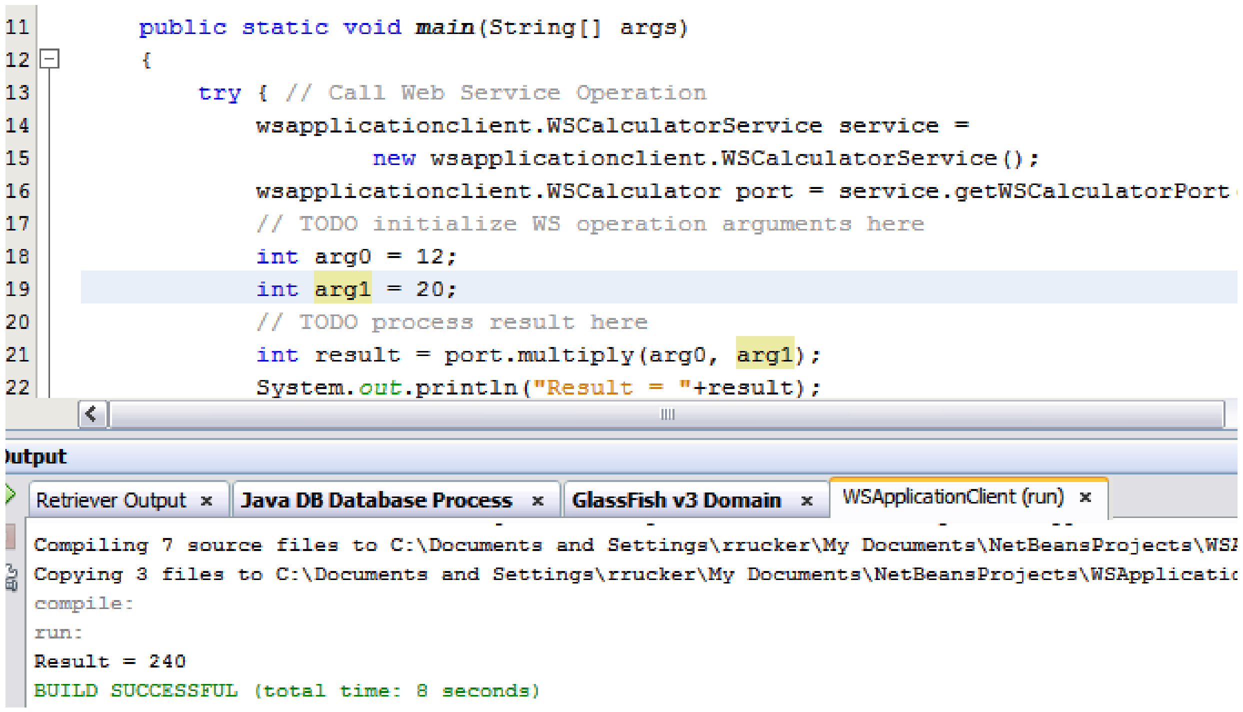
```
/* Main.java Client for the Web Service 'multiply' 2010-06-01 */
package wsapplicationclient;
/**...*/
public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        try { // Call Web Service Operation
            wsapplicationclient.WSCalculatorService service =
                    new wsapplicationclient.WSCalculatorService();
            wsapplicationclient.WSCalculator port = service.getWSCalculatorPort();
            // TODO initialize WS operation arguments here
            int arg0 = 0;
            int arg1 = 0;
            // TODO process result here
            int result = port.multiply(arg0, arg1);
            System.out.println("Result = "+result);
        } catch (Exception ex) {
            // TODO handle custom exceptions here
        }
    }//end method main()
}//end Java class Main
```

Main.java
generated code from referenced web service
you need to assign desired values to arg0 and arg1
this will cause a print out to your console

To test out this client program, you now need to fill in integer values for arg0 and arg1 and simply run the program.

- Edit your Java code and set arg0= 10 and arg1 = 20
- Save the file
- Rt-clk on the *WSApplicationClient* node (that is the project name) and select ‘run’

The output is shown below.



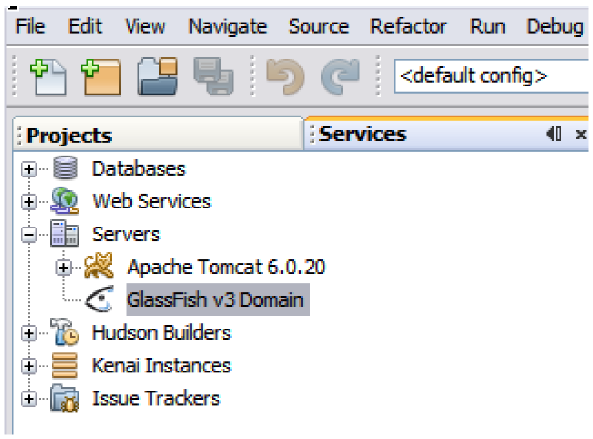
```
    public static void main(String[] args)
    {
        try { // Call Web Service Operation
            wsapplicationclient.WSCalculatorService service =
                    new wsapplicationclient.WSCalculatorService();
            wsapplicationclient.WSCalculator port = service.getWSCalculatorPort
            // TODO initialize WS operation arguments here
            int arg0 = 12;
            int arg1 = 20;
            // TODO process result here
            int result = port.multiply(arg0, arg1);
            System.out.println("Result = "+result);
```

```
Compiling 7 source files to C:\Documents and Settings\rrucker\My Documents\NetBeansProjects\WS
Copying 3 files to C:\Documents and Settings\rrucker\My Documents\NetBeansProjects\WSApplicati
compile:
run:
Result = 240
BUILD SUCCESSFUL (total time: 8 seconds)
```

Starting the Glassfish Server

If the Glassfish server is not running, a web application will fail. So, here is how to start it up.

- Go to menu bar, select ‘Window’ then select *Services*
- Go to Servers
- From that services window, navigate to Glassfish v3 Domain



- Rt-clk and select start -- you are now golden!