Web Services-II Assignment

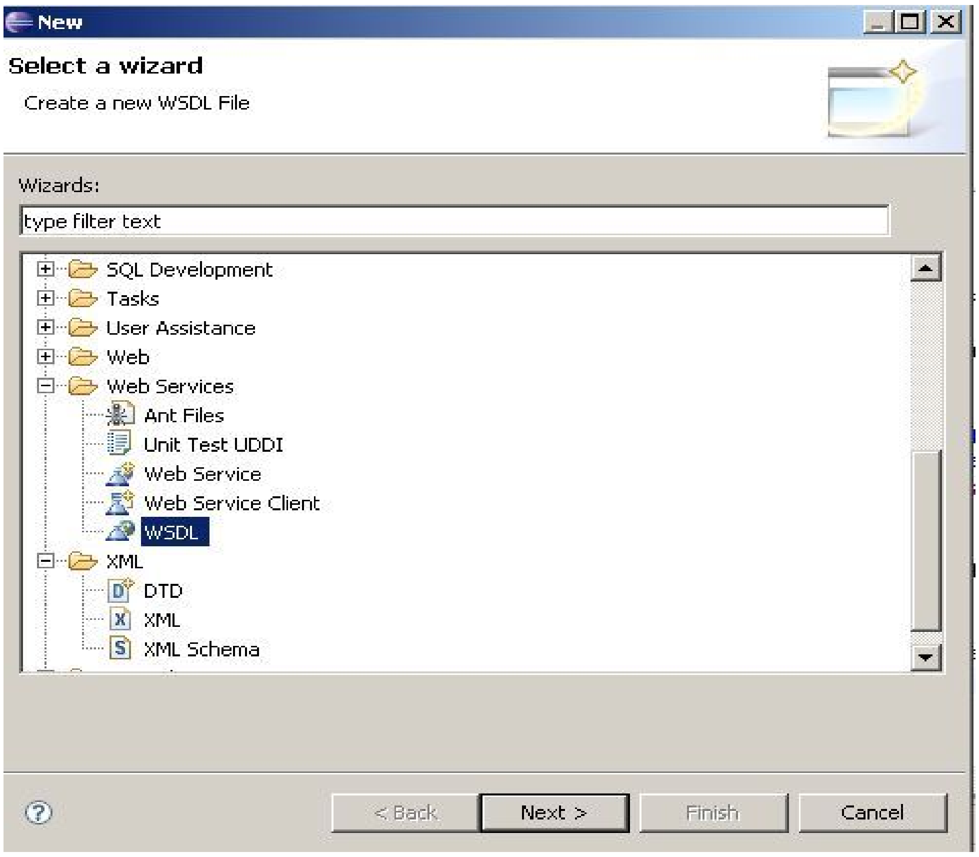
Top Down Approach of Web services Development.

The top down approach (Contract First) takes the contract as the primary artifact. The “contract” for a web service is the WSDL document.

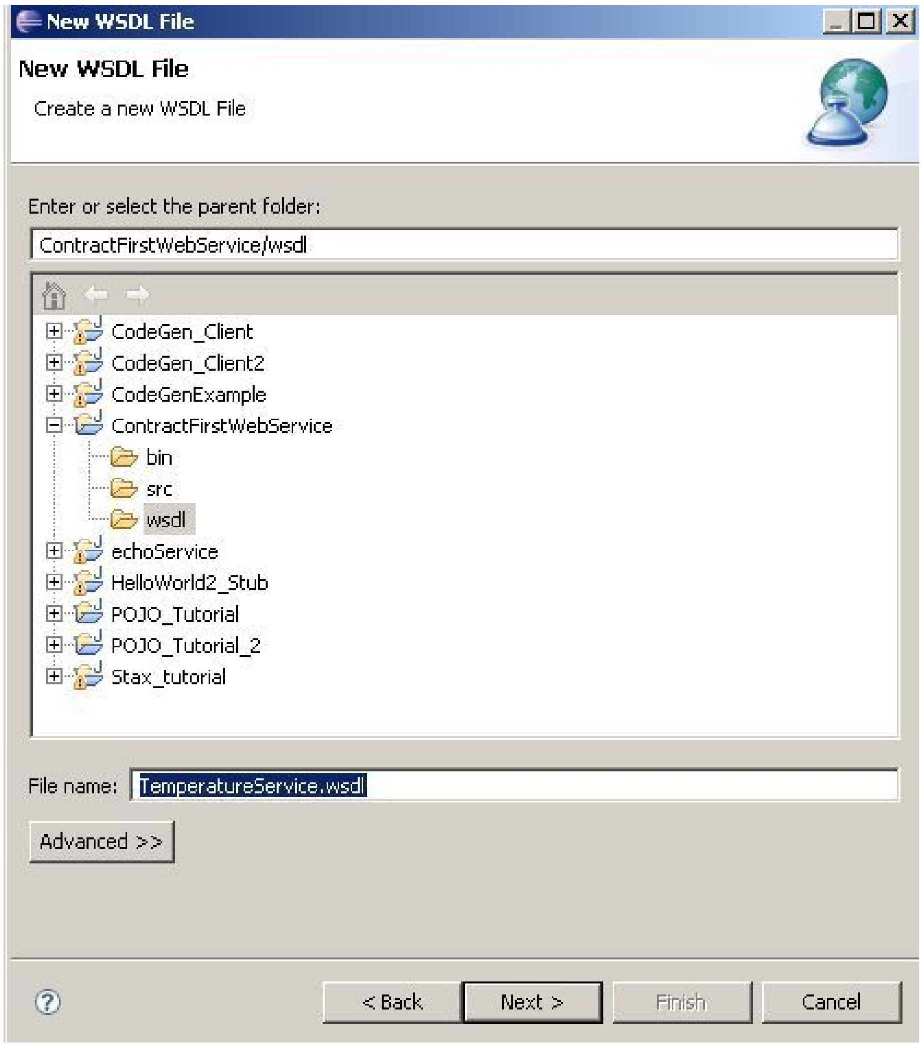
1. Constructing the WSDL File:

We are going to use the WSDL editor provided by Eclipse framework.

I. Create a new java project (ContractFirstWebService)
II. Create a new folder called wsdl
III. To start creating the WSDL file, open the file menu and select new → Other → Web Services → WSDL and press Next



IV. Select the wsdl folder as the parent folder and name the WSDL file as TemperatureService.wsdl and click Next


New WSDL File

New WSDL File

Create a new WSDL File

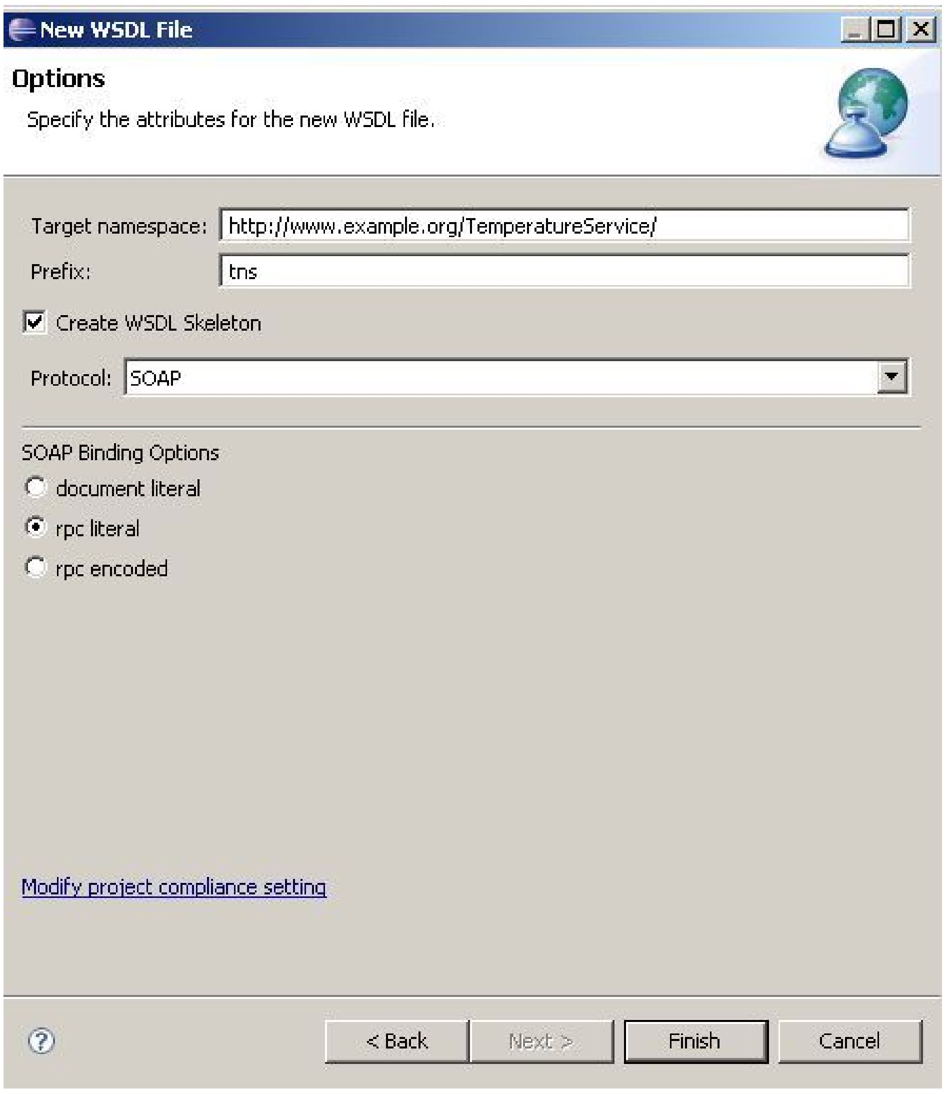
Enter or select the parent folder:

ContractFirstWebService/wsdl

```
CodeGen_Client
CodeGen_Client2
CodeGenExample
ContractFirstWebService
    bin
    src
    wsdl
echoService
HelloWorld2_Stub
POJO_Tutorial
POJO_Tutorial_2
Stax_tutorial
```

File name: TemperatureService.wsdl

V. Keep the default. Just change the soap binding options into RPC literal and click Finish. The WSDL file will be launched by the WSDL editor.



New WSDL File

Options

Specify the attributes for the new WSDL file.

Target namespace: http://www.example.org/TemperatureService/

Prefix: tns

[✓] Create WSDL Skeleton

Protocol: SOAP

SOAP Binding Options

( ) document literal

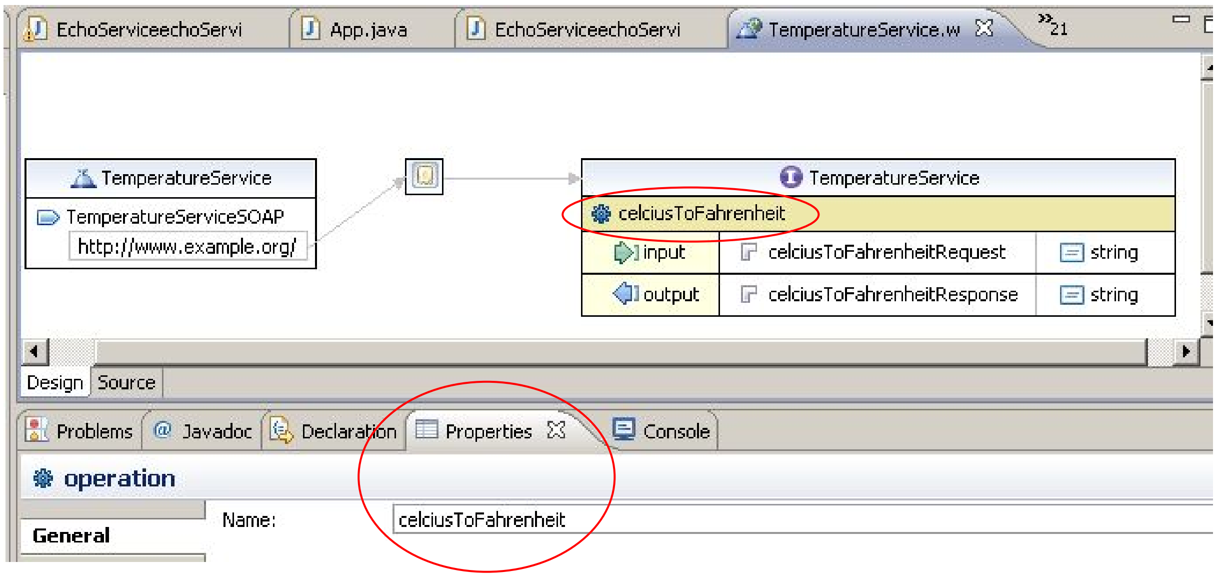
(●) rpc literal

( ) rpc encoded

Modify project compliance setting

< Back | Next > | Finish | Cancel

VI. In the design view, select the operation named **newOperation** under the **TemperatureService** portType and rename it to **celciusToFahrenheit** in the properties view as shown in the figure below.



EchoServiceechoServi | App.java | EchoServiceechoServi | TemperatureService.w

TemperatureService

TemperatureServiceSOAP

http://www.example.org/

TemperatureService

celciusToFahrenheit

input	celciusToFahrenheitRequest	string
output	celciusToFahrenheitResponse	string

Design | Source

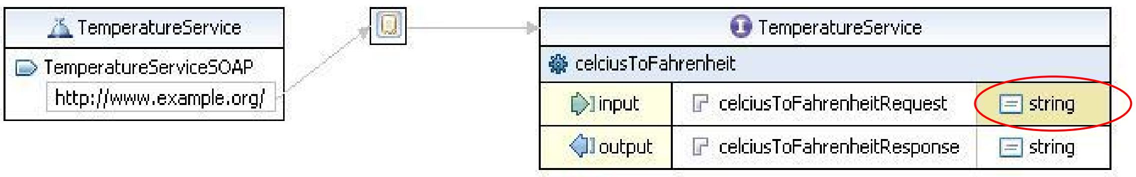
Problems | Javadoc | Declaration | Properties | Console

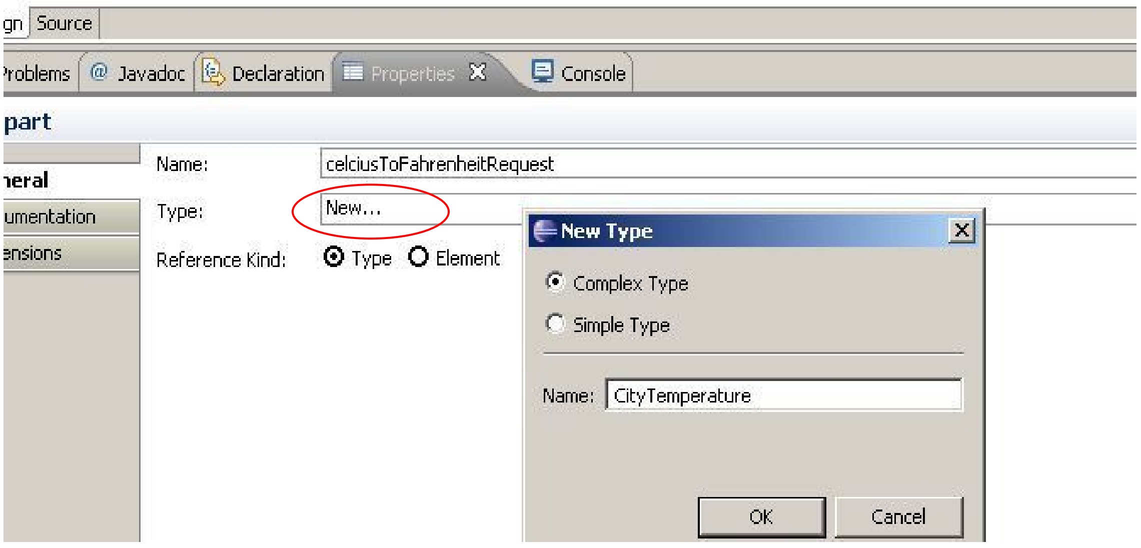
operation

General

Name: celciusToFahrenheit

VII. Suppose that the **celciusToFahrenheit** operation takes a complex type that consisting of two parameters (**city_name, celcius_temperature**) and returns a string "The temperature in city_name is x Fahrenheit degree". Note that the function request and response type are by default Strings. We need to change the request type to a new complex type. This can be done through the properties window as shown in the figure below. Name the new type as **CityTemperature**.

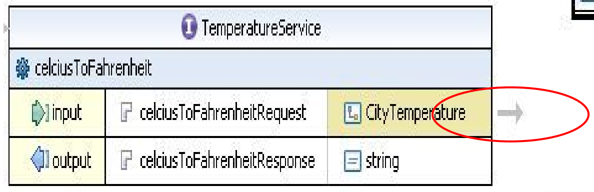


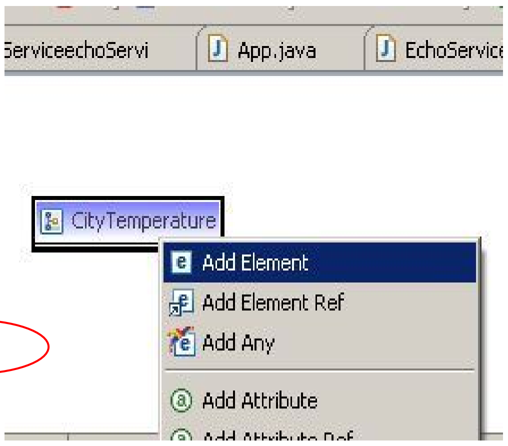


Name: celciusToFahrenheitRequest
Type: New...
Reference Kind: [✓ Type] Element

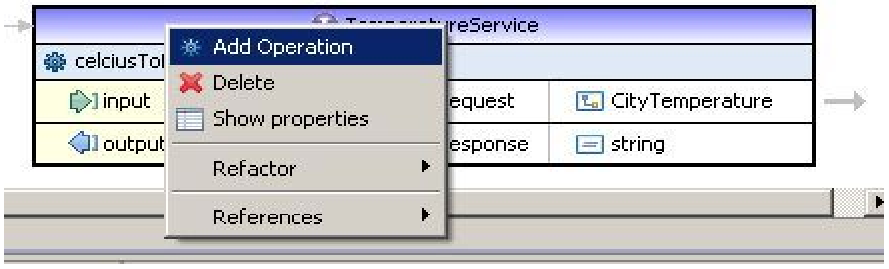
New Type
[✓ Complex Type]
Simple Type
Name: CityTemperature

VIII. In order to define the structure of our new complex type, click on the arrows beside the new type (see the lift side of the figure below). The type will be open in a schema window where you will be able to add new elements as shown in the right side. Add two elements, **cityName (string)** and **temperature (double)**. Finally save and close the schema window.

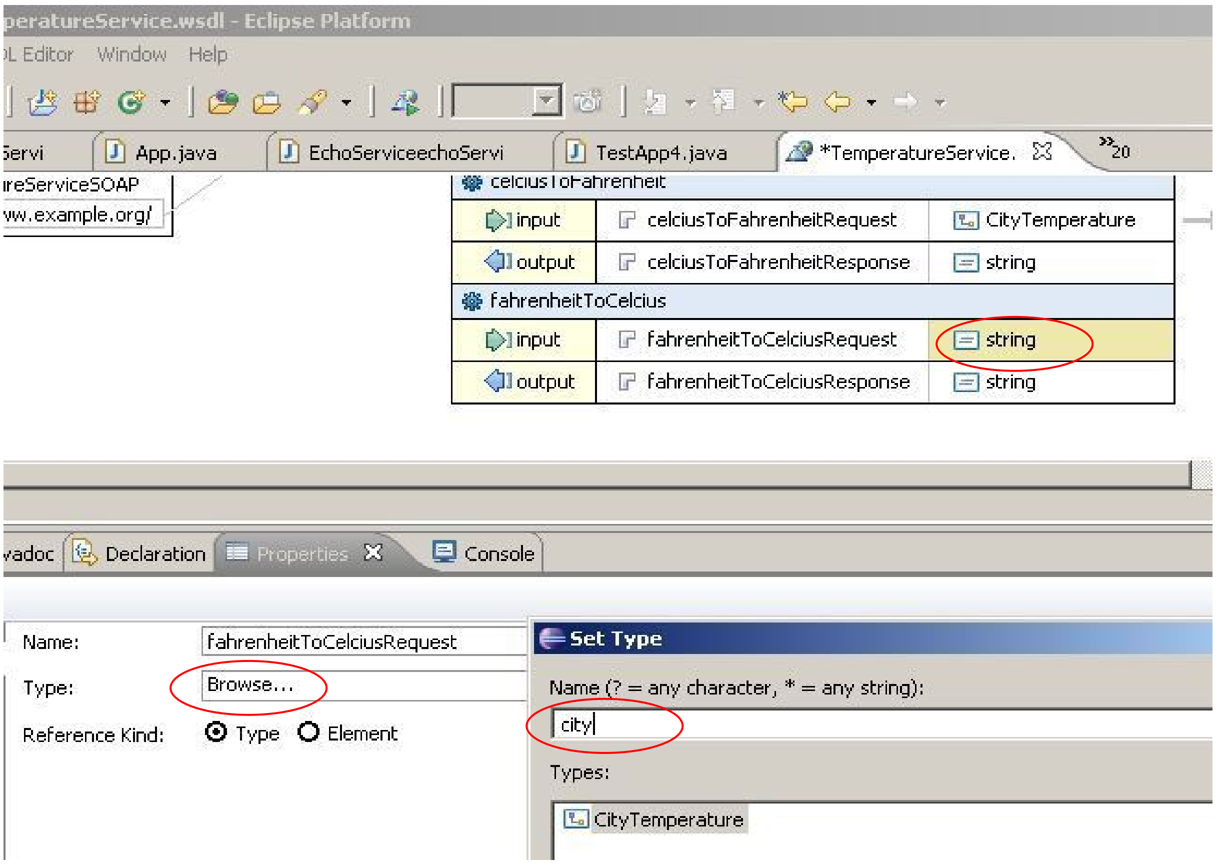




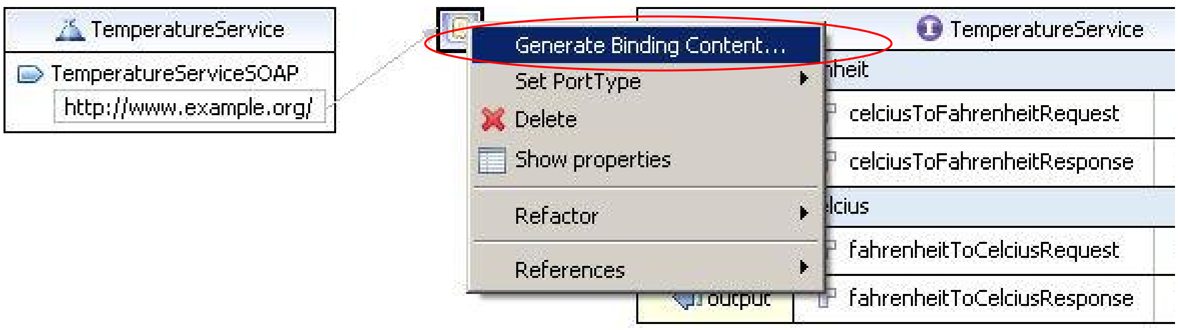
IX. We would like to add another operation **fahrenheitToCelcius**. You can do this by right clicking on the temperature service PortType and selecting the **Add Operation** command.



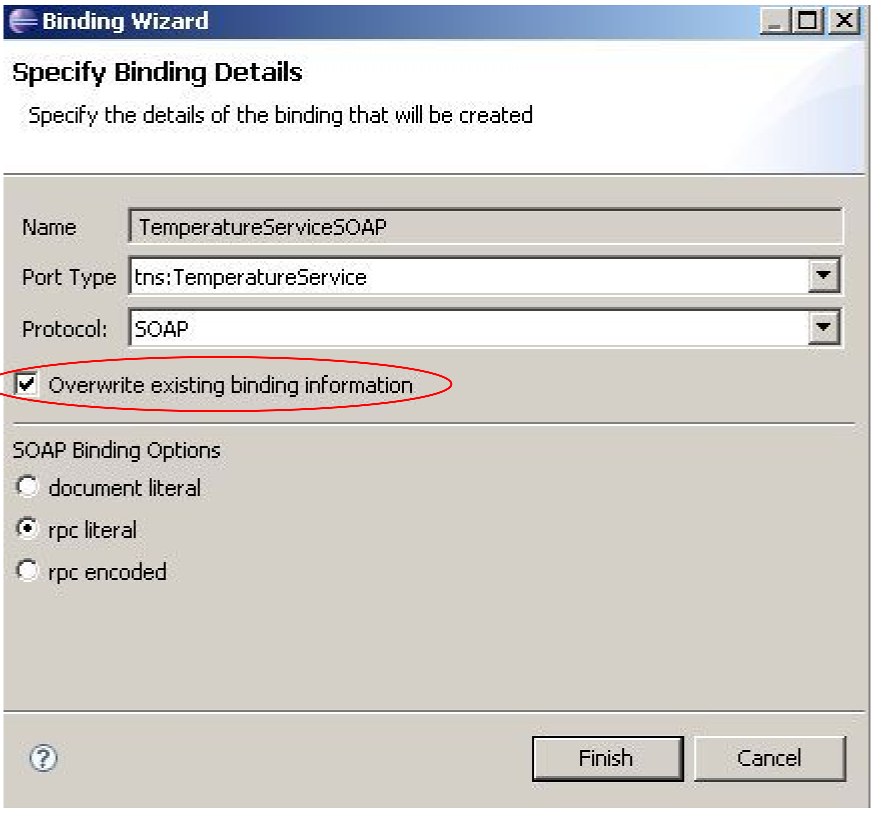
X. Again we need to change request type of the new operation to CityTemperature complex type. This can be done as shown in the figure below.



XI. Finally and in order to update the binding information with our changes, select the binding (the square in the middle), right click and select **Generate Binding Content…**



XII. In the binding wizard, select **Overwrite existing binding information.** Click finish and save the WSDL file.


Binding Wizard

Specify Binding Details

Specify the details of the binding that will be created

Name: TemperatureServiceSOAP

Port Type: tns:TemperatureService

Protocol: SOAP

[✓] Overwrite existing binding information

SOAP Binding Options

- ○ document literal
- ● rpc literal
- ○ rpc encoded

XIII. Finally we have to make sure the WSDL file is valid. To do this, we right-click the file and select **Validate**

2. Generating the Service Skeleton.

To be able to do this step, you have to add the AXIS2_HOME/bin to the PATH environment variable.

I. Open a command prompt and go to the wsdl directory where you store the WSDL file. (inside your workspace)

II. Run the following command:

```
wsdl2java –uri temperatureservice.wsdl –o ../ -ss –sd
```

- **-uri** : specify the location or the URL of our WSDL file.
- **-o** : specify the output location (here we specify the parent directory which refers to our eclipse project's directory.
- **-ss** : indicates that we need to generate the server side code. (service skeleton).
- **-sd** : to generate the service descriptor (services.xml file) along with the service code.

Three artifacts will be visible in the output location:

- **build.xml** : ant file which can be used to generate the required aar file.
- **src dir** : includes the source files (service skeleton)
- **resources dir** : contains two files, the WSDL file for the service, and the services.xml file. These files need to go into the META-INF directory of the aar file.

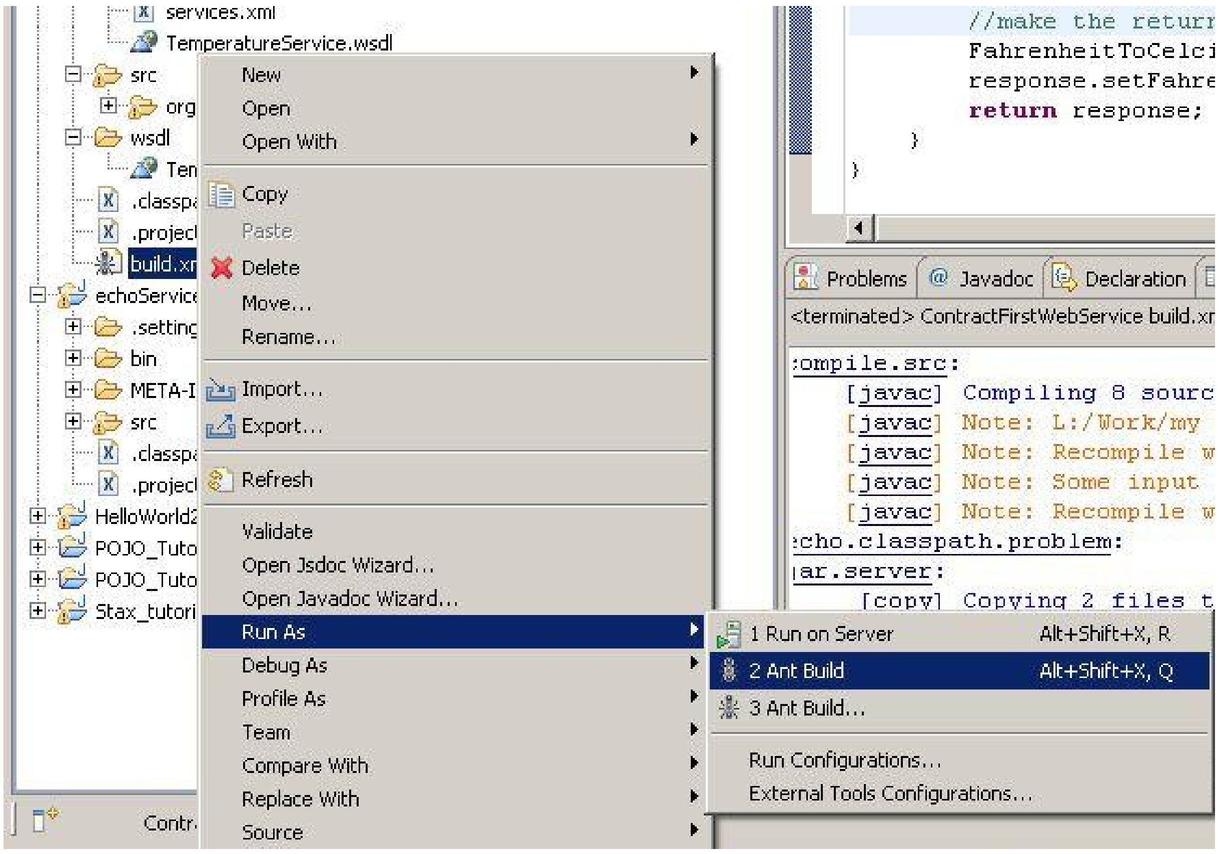
III. Refresh your eclipse project to see the generated files. The code contains errors. To solve this problem, you have to update the build path of your project by adding the jar files located in AXIS2_HOME/lib

IV. The skeleton will be named as **TemperatureServiceSkeleton** to match the service name. For each operation in the WSDL file, a corresponding method will be generated in the skeleton class. The following code shows an implementation for one method:

```
public CelciusToFahrenheitResponse celciusToFahrenheit(
                                   CelciusToFahrenheit celciusToFahrenheit)
{

//get the input data
//1. city name
String cityName= celciusToFahrenheit.getCelciusToFahrenheitRequest().getCityName();
//2. celcius temperature
double celciusTemperature=
              celciusToFahrenheit.getCelciusToFahrenheitRequest().getTemperature();
//3. compute fahrenheit temperature
double fahrenheitTemperature= (9.0/5.0)*celciusTemperature+32;
//make the return object
CelciusToFahrenheitResponse response=new CelciusToFahrenheitResponse();

response.setCelciusToFahrenheitResponse("The temperature in "+cityName +" is
                                      "+fahrenheitTemperature+" fahrenheit degree");
return response;

}
```

V. Implement the second method in the same way.

VI. Use the **build.xml** file as shown in the figure below to construct the aar service file. It will be located in the .\build\lib folder.



```
        //make the return
        FahrenheitToCelci
        response.setFahre
        return response;
    }
}
```

```
<terminated> ContractFirstWebService build.xr
:ompile.src:
    [javac] Compiling 8 sourc
    [javac] Note: L:/Work/my
    [javac] Note: Recompile w
    [javac] Note: Some input
    [javac] Note: Recompile w
:cho.classpath.problem:
jar.server:
     [copy] Copying 2 files t
```

VII. Deploy the services by copying the aar file into the AXIS2_HOME/ repository/services. Then start the Axis2 server from AXIS2_HOME/ bin/Axis2server.bat

VIII. Go to the link http://localhost:8080/axis2/services/, you should find your service listed in the services list. Congratulation you are done ☺

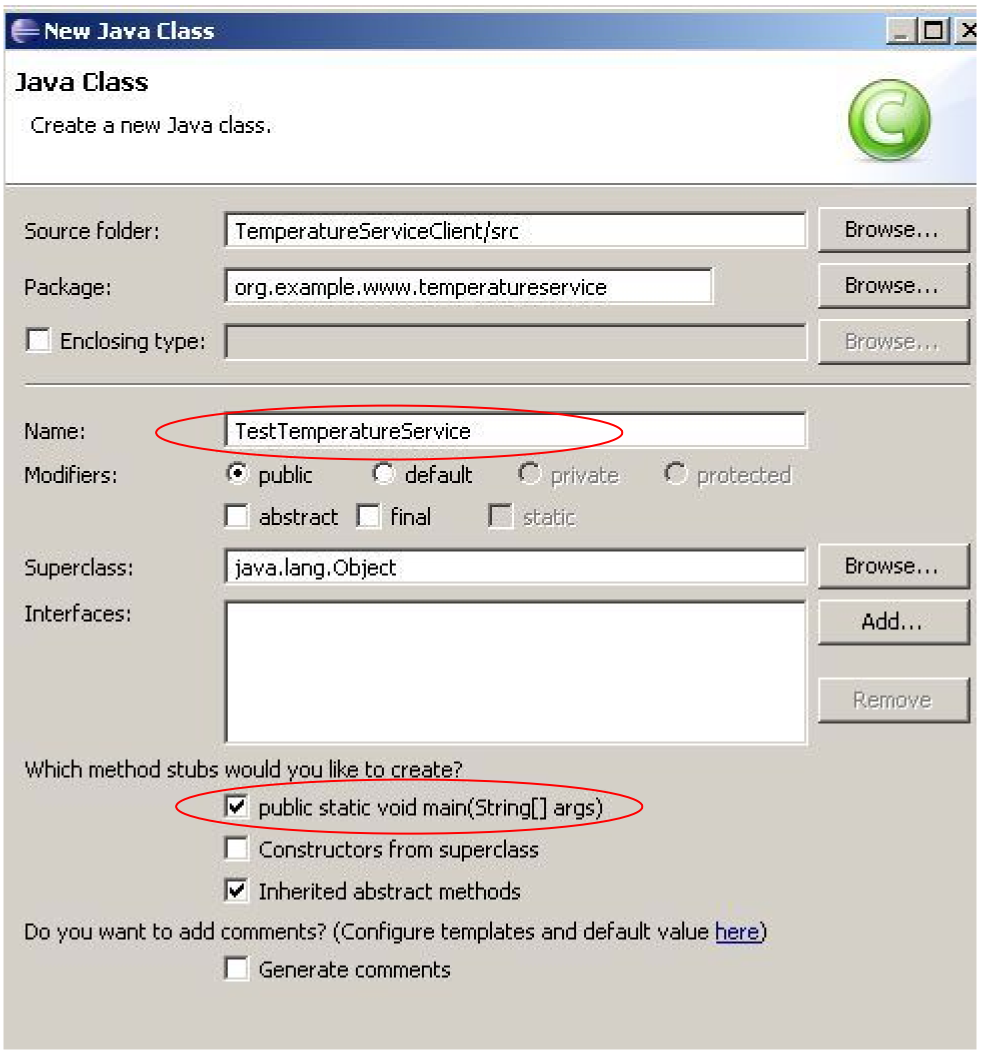
3. Generating Client-Side Code

The client-side code (service stub) generation tool can be used to create client for any remote WSDL file in order to invoke its underlying service.

I. Create a new java project "**TemperatureServiceClient**".
II. Open a command prompt and go to the project parent directory.
III. Run the following command:

```
wsdl2java –uri http://localhost:8080/axis2/services/TemperatureService?wsdl
```

IV. Refresh your eclipse project to see the generated files. The code contains errors. To solve this problem, you have to update the build path of your project by adding the jar files located in AXIS2_HOME/lib
V. The stub class is called **TemperatureServiceStub,** if you open it, you will find it very difficult to read. You don't have to read it; we are not going to edit the stub. Rather we will use it to invoke the remote service.
VI. Create a new class "TestTemperatureService" which contains a main function.


New Java Class

Java Class

Create a new Java class.

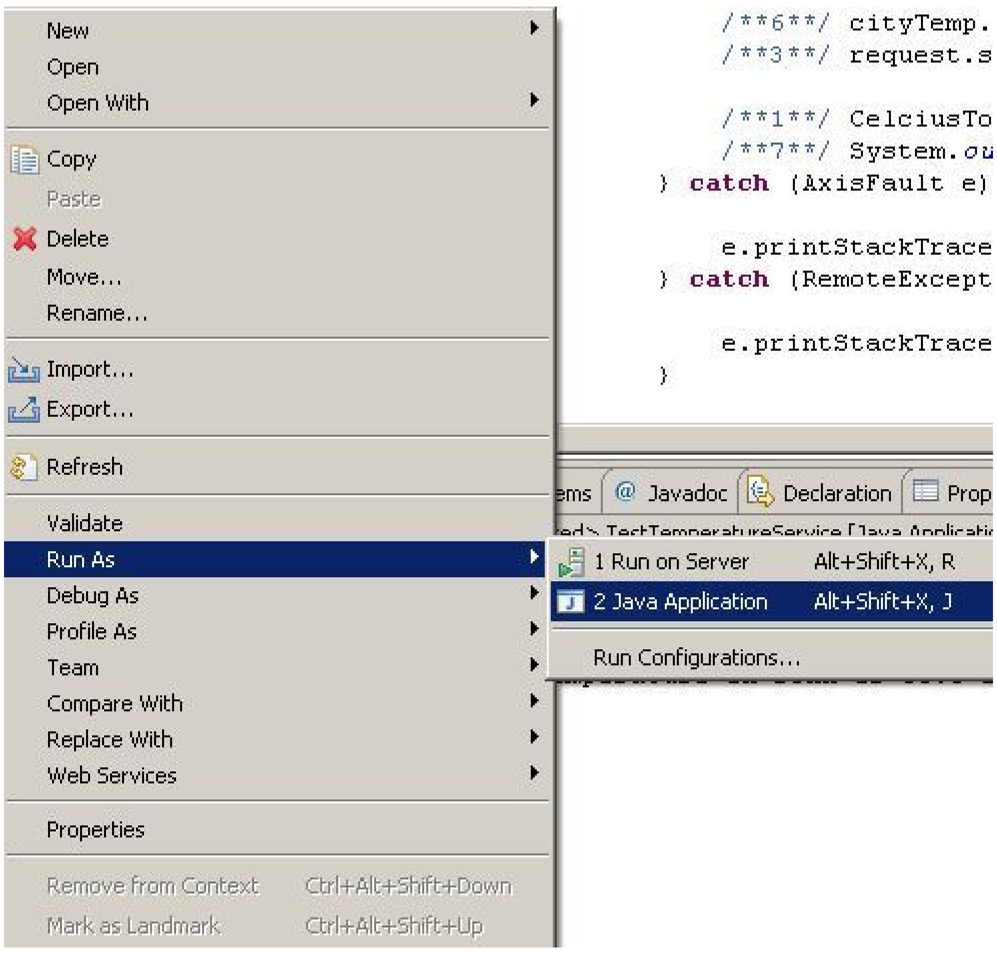
Source folder: TemperatureServiceClient/src
Package: org.example.www.temperatureservice
[ ] Enclosing type:
Name: TestTemperatureService
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

VII. Add the following code to main method:

```
try {
        //create a new stub
        TemperatureServiceStub stub=new TemperatureServiceStub();
        //create CelciusToFahrenheit Object
        /**2**/ CelciusToFahrenheit request=new CelciusToFahrenheit();
        //create a CityTemperature object and initialize it.
         /**4**/ CityTemperature cityTemp=new CityTemperature();
        /**5**/ cityTemp.setCityName("Bonn");
        /**6**/ cityTemp.setTemperature(10);
        //try to set the request,
        //you need a CityTemperature object
         /**3**/ request.setCelciusToFahrenheitRequest(cityTemp);
        //try to call the celciusToFahrenheit method
        //you need a CelciusToFahrenheit object (no. 2)
        /**1**/ CelciusToFahrenheitResponse response=
                                   stub.celciusToFahrenheit(request);
        //print the response string.
        /**7**/ System.out.println(response.getCelciusToFahrenheitResponse());
      }
catch (AxisFault e) {
        e.printStackTrace();
}
catch (RemoteException e) {
        e.printStackTrace();
}
```

VIII. Run the application as shown below and check the final outcome.



```
        /**6**/ cityTemp.
        /**3**/ request.s

        /**1**/ CelciusTo
        /**7**/ System.ou
    } catch (AxisFault e)

        e.printStackTrace
    } catch (RemoteExcept

        e.printStackTrace
    }
```