Assignment 1.1 - Web Service (WDSL/SOAP) - Lab Report

Katerina Chinnappan, Rohit Shaw, Kjell Zijlemaker

Design & Approach

We have chosen Java to implement the web service & client for this part of the assignment. We have used the *Bottom-Up* approach(started off by writing the Java class/methods and then generated the WSDL file from it) to implement the WDSL/SOAP service. Essentially, SOAP means that the URL remains the same for all invocations. The only part that changes are the parameters for the Java methods. We set up the project in the NetBeans IDE because it provides a convenient integration with the GlassFish server. We chose "create web service" when creating the project in NetBeans and this created a nice project structure. Maven is used in NetBeans in order to package the entire project (web service) in a .WAR file. This file is then used to deploy the web service to a server (in our case we went with GlassFish, which is what NetBeans offered.)

Implementation

Web Service - `CalculatorService.java`

We started off by creating a Java class (`CalculatorService.java`) for the web service in the **src /main /java/ package** directory. Since we were implementing a web service, we used the dependency injection `@WebService` right before the class definition. We then proceeded to implement the methods add, sub, mul & div. Just like the `CalculatorService.java` class, each method used a `@WebMethod` dependency injection in order to explicitly state that this is indeed a web method inside a web service. Each method takes in two parameters of type `float` and returns the result from the mathematical operation performed on the two parameters.

After finishing implementing the web service, NetBeans takes care of packaging the project in a .WAR file (the type of file we wish to package our project too is specified in the `pom.xml` file in the root of the project). This .WAR file is then deployed to GlassFish server (through NetBeans) and the web service can be accessed through `http://localhost:8080/calculator_maven/CalculatorServiceService?Tester`

Client - `CalculatorClient.java`

Of course, `http://localhost:8080/calculator_maven/CalculatorServiceService?Tester` this url is a client itself however we wrote our own client to illustrate that we understand how to use our web service. The client folder and Java classes are generated from the WSDL file which was generated in the beginning using the Bottom-Up approach. The command `wsimport -keep -p client calculator.wsdl` generates all the needed Java classes in order to write the client. The generated file `CalculatorServiceService.java` initiates a connection to the web service so in our case we had to change the url to `http://localhost:8080/calculator_maven/CalculatorServiceService?WSD`. The generated file texttttCalculatorService.java describes the web methods. We must create a new `service` object - `CalculatorServiceService service = new CalculatorServiceService();`. We then create the `calc` object - `CalculatorService calc = service.getCalculatorServicePort();` This specifies which port number to use for the protocol. We then can proceed to write our client by calling the web methods described in `CalculatorService.java` using the `calc` object. We decided to provide an option for the user to test the web service using the client in an interactive and non-interactive mode. The non interactive mode simply invokes the web methods with hard-coded numbers and returns the output. The interactive mode provides a menu with the list of mathematical operations and gives the option to the user to chose which operation to perform on the chosen numbers.