Télécom Saint-Étienne
Web Services
TP n°2 - 3h00

Enseignant Syed Gillani

Remarks

- You need to form groups of two students (binôme).
- A binôme should write a report containing the answers of questions and exercises listed in this document and send them to syed.gillani@univ-st-etienne.fr.
- DEADLINE: 19/10/2016 À 20h

Objectives

- Write a SOAP web service and test it.

1 Install Java EE and Run GlassFish

Eclipse EE is an IDE for Java EE (Java Platform Enterprise Edition). The platform provides an API and runtime environment for developing and running the enterprise software, including network and web services.

Java EE can be downloaded from the Oracle web site (as shown below).

1. Go to this link, download the .zip file and extract it.
2. You should get a folder named glassfish4. **GlassFish** is an open-source application server project[1].
3. From the command line (cmd) go to the extracted glassfish4/bin folder.
4. In order to start the server, type the following command **./asadmin start-domain**
5. Hopefully, the GlassFish server is now started. To test it go to internet browser and type **localhost:4848/**. The port 4848 is the default port of GlassFish.

Now, you've installed Java EE and you have a server (GlassFish) running on your computer.

2 Set GlassFish in Eclipse EE

Eclipse EE includes most of the useful components and techniques that can be used to develop web services and web applications. Therefore we will download it also since it is very useful. Moreover, we will integrate GlassFish into Eclipse EE and thus we can run GlassFish from Eclipse and not from the terminal.

2.1 Downloading Eclipse EE (if you haven't got it already)

1. Go to this link and download the eclipse version that corresponds to you operating system.
2. Go to the file you downloaded and launch Eclipse.

1. For those of you who know Apache Tomcat, GlassFish is similar but more powerful and complete.

2.2 Setting a GlassFish Server in Eclipse

1. From Eclipse go to `Window` ►`servers`. A new window will appear in the Eclipse showing that there is no server installed.
2. Click on the link to create a new server. You need to select the server type. Probably, you will not find GlassFish type. That's normal because you need to install add its plugin to the Eclipse.
3. Click on `Download additional server adapters` link and a list of available servers will appear. Choose **GlassFish Tools** from the list. ►`Next` ►accept the license agreement ►finish (If it doesn't work then: Go to `Help` ►`Eclipse Marketplace` ►Search `GlassFish` ►you will see `GlassFish Tools` ►Select appropriate one and install it).
4. The plugin will be downloaded. Now restart Eclipse.
5. Now add a new server (up till now, you downloaded the plugin but you did not add the server).
6. Choose the server type as `GlassFish` ►GlassFish4.0 ►`Next`.
7. Choose the JRE 1.7
8. Specify the location of GlassFish directory (glassfish4) that you have already installed (use `Browse`) and hit `Next` then `Finish`.
9. You have the server added to Eclipse and it appears in the window server. Right click on the server name and hit start. The server has now started.
10. Right click on the server and hit `GlassFish` ►`View Admin Console`. Now you can see you **localhost:4848/** from Eclipse.

3 Create The SOAP Service

From Eclipse EE, do the following:

1. `File` ►`Project` ►`Web.`►`Dynamic Web Project`. **Do not choose New Web Service project!**
2. Name the project **CurrencyConvertor**.
3. Choose GlassFish 4.0 as the `Target runtime`.
4. For Configuration: leave it as default Configuration for GlassFish ►hit `Finish`. This project is a web project. However, it does not contain web pages. We will add a web page.
5. In the Project Explorer, right click on the project name (CurrencyConvertor) ►`New` ►`JSP Page`.
6. Call the new file `index.jsp` ►hit `Finish`.
7. From the Project Explorer open the folder `WebContent`. You will find the page `index.jsp`. Double click on it.
8. Add: `<h1> Welcome to Money Converter</h1>` to the body of `index.jsp`.
9. To test the page on the server, right click on the index.jsp (from the Project Explorer) ►`Run as` ►`Run on Server`. The page should be displayed correctly.
10. Let us now add the business logic. Add a new **Java Class** to the project and specify the package as *org.yourname.webservices*. Call the class MyCurrencyConvertor.
11. In this class add a method *getCurrenciesList()* returning a list of String containing the names of the currencies that your service can handle (i.e. Dollar, Euro, Yen). This method relies on another method called initializeList() that initialize the list if it is empty.
12. Up till now, we have a web project containing a web page and a Java class describing the business logic. **But that is not a web service**. To transform your class a web service: **add the annotation *@WebService before the class declaration.*** Do not forget to make the necessary import from `import javax.jws.WebService`.

13. To deploy the web service, right click on the project name ▶`Run As` ▶`Run on Server`. **You will get 404 error page, do not worry. This is because you do not have a root page.**
14. You've created a SOAP web service. Still, you need to test it.

4 Testing a Local Web Service

To test your web service, you do not need to write a java client. GlassFish provides an integrated tool to do this.

1. Open your internet browser and go to GlassFish Admin Page.
2. On the right column choose `Applications`. This will show you the web services deployed on your GlassFish Server.
3. You should find you CurrencyConverter service. Click on it.
4. You will get a detailed description of the service. Scroll down you find the name of the class you annotated as a web service (i.e. myCurrencyConverter). Click on `View The Endpoint`.
5. Finally, a new page will appear. It contains two links the first is the tester and the second is the WSDL. Click on the WSDL link. You see how GlassFish automatically generated a WSDL for you web Service. Do not forget to take a look.
6. Now go to the Tester link. Open it. You will see the Methods available in your class. Click on the button that invokes *getCurrenciesList()*.
7. You will get a list of the currencies. Do not forget to take a look at the SOAP request and SOAP response generated by the GlassFish server and the test client. Note how the request contains no arguments since the method **getCurrenciesList** takes no argument. Also take a look at the SOAP response returned by our web service.

Question 1: Put both the WSDL, the SOAP request and response in your report.

5 Adding Methods Taking Parameters

5.1 The Convert Method

Now add to your class a method **convert (String source, String destination, double amount)** that converts a given amount of money from the source currency to the destination currency.

From Google, find the conversion prices between the three currencies and write the method. Run the class on the server and then See the WSDL of the method and test it with different values.

Question 2: Add the new WSDL, the SOAP requests and responses in your report.

5.2 Add New Currency

Write a method that adds a new currency: **addCurrency(String name)** to the list and test it. Usually add methods return a boolean indicating whether the add operation was successful or not. **Question 3: Add the new WSDL, the SOAP requests and responses in your report.**

Imagine that you decided that now one should add a new currency to your service. So you need to *exclude* the addCurrency method. Go to your code, add the annotation **@WebMethod(exclude=true)** before this method definition. This annotation means that the method is no longer a method that can be invoked over the web.
Test it. Check the new WSDL.

Question 4: Compare the new WSDL with old one, what are the differences?

5.3 Using User-defined Types

Up till now, all the methods that you have developed have simple types as input and return types. Nevertheless, SOAP is capable of exchanging objects belonging to user-defined classes. This exercise shows how this is done.

1. Write a new class called Currency. This class contains three attributes:
   (a) String : name, the name of the currency (e.g. "Euro").
   (b) String :country, the name of the countries in which the currency is used (e.g. "EU" for Euro).
   (c) int yearAdopted, the year in which this currency was adopted for the first time (e.g. 1999 for Euro).
2. Put these attributes as private, and add public setters and getters for them.
3. Create a new Java class called **MyCurrencyConverter2**. Copy paste the content of the class MyCurrencyConverter. Do not forget to copy the annotation **@WebService**.
4. What we need to modify is only the **getCurrenciesList()** method. It should become: `public List<Currency> getCurrenciesList()`. We do not need other methods in this class to be Webmethods. Therefore, exclude them by adding the annotation @WebMethod(exclude=true) before each one of them.
5. Run the new service on the server and test it using the tester.
6. From the tester call the method getCurrenciesList().
   Question 5: What to you see in the SOAP response? Compare it with the one generated by the old getCurrenciesList() what are the differences? Do not forget to add the SOAP response to you report.

7. Open the WSDL of the new service[2]. In the WSDL, you find tag called `<xsd:schema>`. This tag indicates the XML Schema (the data types) used in this Service. There is an attribute named `schemaLocation`. Copy the link and open it in the browser. You will get and XML Schema defining the type used in the service.
   Question 6: Add this XML schema to your report. Give a list of the types used by you service. What do you think? Please give your comments about the types you found.

Important Remarks

– Compress the project as a .zip file, attach it with the answers (pdf file) and send it to the email address mentioned above. Please put "WS-TD" in the subject (title) of the email.

2. MyCurrencyConverter2