Olga Melekhova

TP 3
CONTRACT LAST APPROACH FOR SOAP WEB SERVICES WITH JAX_WS

Web services (from wikipedia):
A Web service is a method of communication between two electronic devices over World Wide Web. A web service is a software function provided at a network address over the web or the cloud; it is a service that is "always on" as in the concept of utility computing.
The W3C defines a "Web service" as:
[...] a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.
XML web services use Extensible Markup Language (XML) messages that follow the SOAP standard and have been popular with the traditional enterprises. In such systems, there is often a machine-readable description of the operations offered by the service written in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP endpoint, but it is a prerequisite for automated client-side code generation in many Java and .NET SOAP frameworks (frameworks such as Apache Axis2, Apache CXF, Spring, gSOAP being notable exceptions). Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a web service.

Lab work objectives : Interoperability regarding web services allows using different languages, operating system..., on both sides (server and client). Nevertheless, here, only JAX_WS is used: a Java library that makes Web Service development easier.

Required material :
- Java JDK only version 1.6 - 1.8, ... (not JRE)
- Eclipse version JEE developer
- Gradle
- JAX WS
- Template: https://github.com/olgamelekhova/soap_tpl

Step 1. Create a Gradle project.

JAX WS can be managed (download, integrated into the project...) automatically with Gradle:
- use provided template and make import as an existing Gradle project
- use the following build.gradle file (see: https://github.com/olgamelekhova/soap_tpl/blob/master/build.gradle)

Step 2: Server side code.

Three files are required:
- a java interface for the web service definition (a list of methods)
- the class that implements the interface
- a main program that starts the web server and publishes the web service.

2.1 A java interface for the web service definition, see a list of methods :
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/ws/CarService.java

2.2 The class that implements the interface.
Annotations (WebService and WebMethod) are used by JAX WS to generate automatically additional code for the

web service publishing (use as import javax.jws...). The class implementing the web service can be deduced from the interface (take care at the package name in the WebService annotation).
Implement the followng methods from the previous paragraph:
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/ws/CarServiceImpl.java:

- getCar - get car by its plate number
- rentCar - find car by its plate number and rent if it's available
- getBackCar - find car by its plate number and get it back if it was rented before

2.3 A main program that starts the web server and publishes it:
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/endpoint/ServicePublisher.java
Publishing a web service is very simple with JAX_WS :
Endpoint.publish("http://localhost:8080/ws/carservice", new CarServiceImpl());

It will start a web server and install the web service. The service is accessible at the given URL with auto-generate WSDL description: http://localhost:8080/ws/carservice?wsdl.

Step 3: Client side coding.

JAX WS is able to parse remotely the WSDL and to generate helper classes that make client side code writing easier. Indeed, the JDK includes a command wsimport.

Generate helper classes using wsimport tool:

- wsimport -s . http://localhost:8080/ws/carservice?wsdl
- add generated files to the directory:
  https://github.com/olgamelekhova/soap_tpl/tree/master/src/main/java/com/example/client/service
- check package correctness of generated classes (ex.:package com.example.client.service;).

Then use
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/client/SoapClient.java file to execute methods and make requests to your web service.

Launch your web server:
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/endpoint/ServicePublisher.java
Launch your client:
https://github.com/olgamelekhova/soap_tpl/blob/master/src/main/java/com/example/client/SoapClient.java

NOTE: Optional step to configure PATH system variable if your wsimport executable is not visible: https://www.java.com/en/download/help/path.xml
NOTE (FOR ECLIPSE): In order to run web server and client in parallel mode, open 2 console using the tools provided by IDE.
NOTE: See official documentation for JAX_WS if you have some problems.