LAB 4 -SOAP

NB : This pom should work in java 8 (lab vms).
WSDL is an xml file that defines the web service.
Its divided in **Types**, **Message**, **portType**, **Binding**, **Service.**

- **Service:** In our case we have 1 service, whose name is "WSImplService", and whose binding is on soap at the address "localhost:8080/WSInterface"

```
<wsdl:service name="WSImplService">
  <wsdl:port binding="tns:WSImplServiceSoapBinding" name="WSImplPort">
    <soap:address location="http://localhost:8080/WSInterface"/>
  </wsdl:port>
</wsdl:service>
```

- **Binding:** Basically tells us the type of marshalling (doc style) that we are using for all the operations of the web service.

```
<wsdl:binding name="WSImplServiceSoapBinding" type="tns:WSInterface">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getStudents">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="getStudents">
      <soap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getStudentsResponse">
      <soap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="hello">
    <soap:operation soapAction="" style="document"/>
    <wsdl:input name="hello">
      <soap:body use="literal"/>
    </wsdl:input>
```

- **PortType:** Tells us how the messages are exchanged in each operation

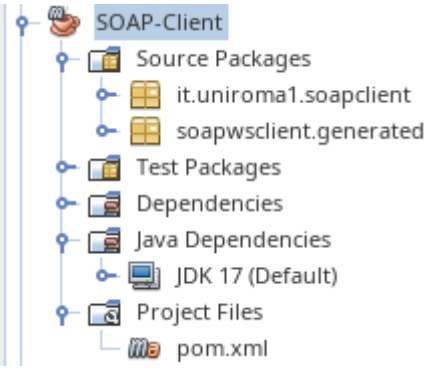
```
<wsdl:portType name="WSInterface">
  <wsdl:operation name="getStudents">
    <wsdl:input message="tns:getStudents" name="getStudents"> </wsdl:input>
    <wsdl:output message="tns:getStudentsResponse" name="getStudentsResponse"> </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="hello">
    <wsdl:input message="tns:hello" name="hello"> </wsdl:input>
    <wsdl:output message="tns:helloResponse" name="helloResponse"> </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="helloStudent">
    <wsdl:input message="tns:helloStudent" name="helloStudent"> </wsdl:input>
    <wsdl:output message="tns:helloStudentResponse" name="helloStudentResponse"> </wsdl:output>
  </wsdl:operation>
</wsdl:portType>
```

- **Message:** contains the name of the message and its parts.

```
</wsdl:message>
<wsdl:message name="helloStudentResponse">
...
</wsdl:message>
<wsdl:message name="getStudents">
...
</wsdl:message>
<wsdl:message name="helloStudent">
...
</wsdl:message>
<wsdl:message name="helloResponse">
...
</wsdl:message>
<wsdl:message name="getStudentsResponse">
  <wsdl:part element="tns:getStudentsResponse" name="parameters"> </wsdl:part>
</wsdl:message>
```

- **Types**: Recursively describes the types returned by the messages.

Client



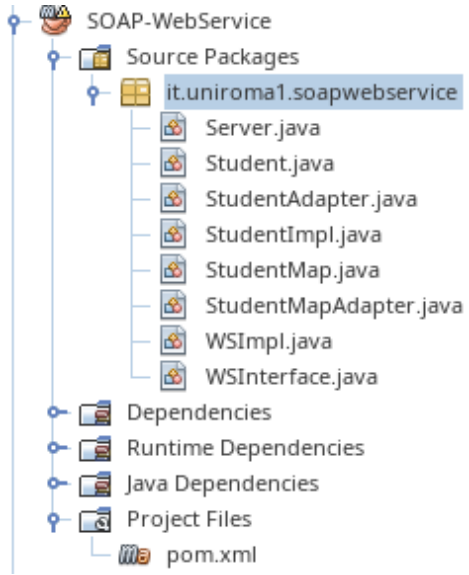
On the client this wsdl is used to automatically generate a class for each message and a class for each "type" (also the pom dependencies are important).
Once the code is generated, in a main class (Client . java) we can put our app logic

```
public class Client {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.setName("Peppino");
        Client.helloStudent(s1);
        Student s2 = new Student();
        s2.setName("Marina");
        Client.helloStudent(s2);

    System.out.println("... and now recovering the whole StudentMap ...");
        List<StudentEntry> result = Client.getStudents().getEntry();
        for (int i = 0 ; i<result.size(); i++){
            System.out.println(i + ": " + ((StudentEntry)result.get(i)).getStudent().getName());
    }

    }

    private static StudentMap getStudents() {
        WSImplService service = new WSImplService();
        WSInterface port = service.getWSImplPort();
        return port.getStudents();
    }

    private static String helloStudent(Student arg0) {
        WSImplService service = new WSImplService();
        WSInterface port = service.getWSImplPort();
        return port.helloStudent(arg0);
    }
}
```

Notice how we can declare the generated types of objects and use their methods (Student and setName) and we can call the methods (or operations to be precise) by calling them on the tight objects and redefining them like above.

OLD, NEED TRO CHECK: right click on package -> new -> other -> WebServiceClient (specify url)

Server



Add dependencies e plugin to pom.xml -> clean and build.

WSInterface. java declares the methods that the app supports (only the declaration). Note the "@WebService". This is to create the interface.
Also add the "@XmlJavaTypeAdapter" to note all the get/set methods that use a non-standard return type. This declares the need for an Adapter class (which has to be implemented)

```java
package it.sapienza.softeng.soapws;

import java.util.Map;
import javax.jws.*;
import javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter;

@WebService
public interface WSInterface {

    public String hello(String name);

    public String helloStudent(Student student);

    @XmlJavaTypeAdapter(StudentMapAdapter.class)
    public Map<Integer, Student> getStudents();
}
```

```java
package it.sapienza.softeng.soapws;

import java.util.LinkedHashMap;
import java.util.Map;
import javax.xml.bind.annotation.adapters.XmlAdapter;

/**
 *
 * @author biar
 */
public class StudentMapAdapter extends XmlAdapter<StudentMap, Map<Integer,Student>>{

    public StudentMapAdapter(){}
    @Override
    public Map<Integer,Student> unmarshal(StudentMap v) throws Exception {
        Map<Integer,Student> boundMap = new LinkedHashMap<Integer, Student>();
        for (StudentMap.StudentEntry studentEntry : v.getEntries()) {
            boundMap.put(studentEntry.getId(), studentEntry.getStudent());
        }
        return boundMap;
    }

    @Override
    public StudentMap marshal(Map<Integer, Student> boundMap) throws Exception {
        StudentMap valueMap = new StudentMap();
        for (Map.Entry<Integer, Student> boundEntry : boundMap.entrySet()) {
            StudentMap.StudentEntry valueEntry = new StudentMap.StudentEntry();
            valueEntry.setStudent(boundEntry.getValue());
            valueEntry.setId(boundEntry.getKey());
            valueMap.getEntries().add(valueEntry);
        }
        return valueMap;
    }
```

- **WSImpl .java:** implements the endpoint's interface. Note the @WEbService(endpointInterface="...").

```java
package it.sapienza.softeng.soapws;

import java.util.LinkedHashMap;
import java.util.Map;
import javax.jws.WebService;

@WebService(endpointInterface = "it.sapienza.softeng.soapws.WSInterface")
public class WSImpl implements WSInterface {

    private Map<Integer, Student> students = new LinkedHashMap<Integer, Student>();


    public WSImpl() {}
    public String hello(String name) {
        return "Hello " + name;
    }

    public String helloStudent(Student student) {
        students.put(students.size() + 1, student);
        return "Hello " + student.getName();
    }

    public Map<Integer, Student> getStudents() {
        return students;
    }
}
```

- **Server. java:** Implements the server that listens on a given url for calls to the ws

```java
package it.sapienza.softeng.soapws;

import javax.xml.ws.Endpoint;

public class Server {

    public static void main(String args[]) throws InterruptedException {
        WSImpl implementor = new WSImpl();
        String address = "http://0.0.0.0:8080/WSInterface";
        Endpoint.publish(address, implementor);
        while(true) {}
        //Thread.sleep(60 * 1000);
        //System.exit(0);
    }
}
```