

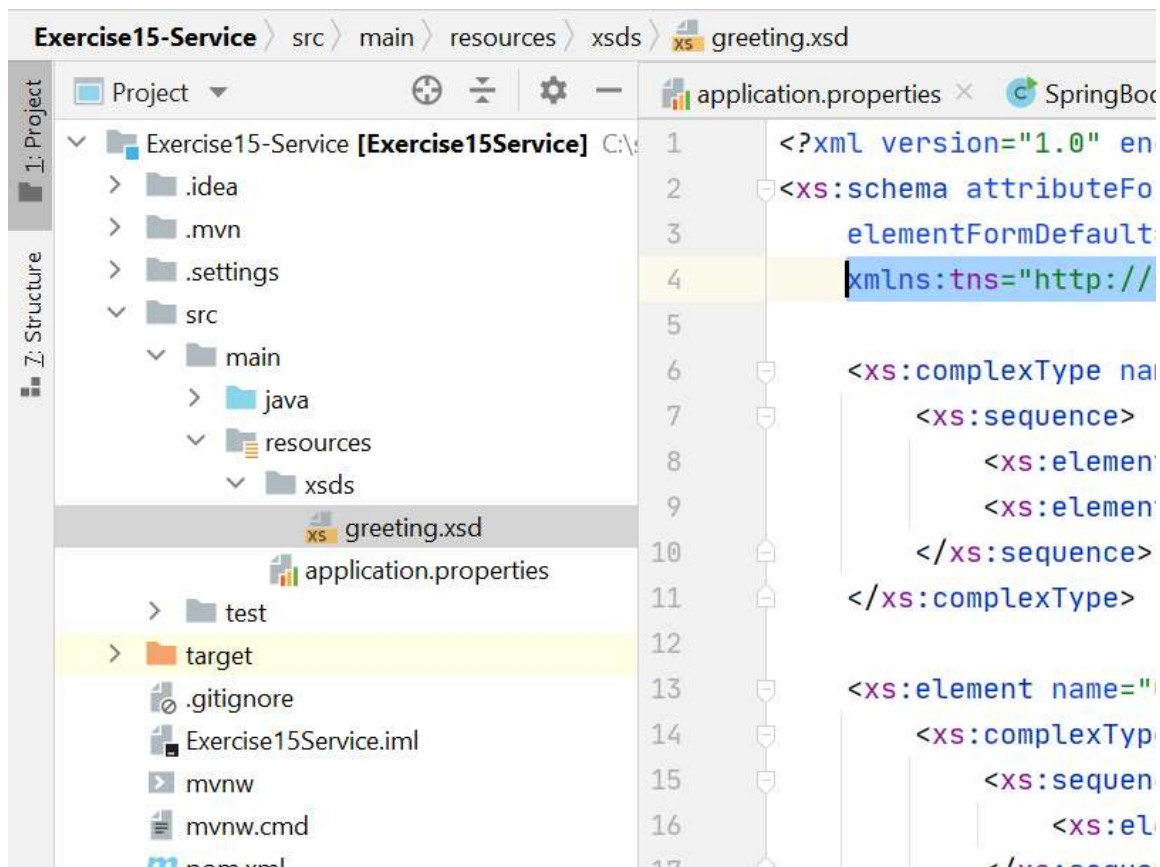
Lab 10

Part A:

Wherever you see the text **Exercise15-Service** in the screenshots, you should see **Lab10-Service** in IntelliJ

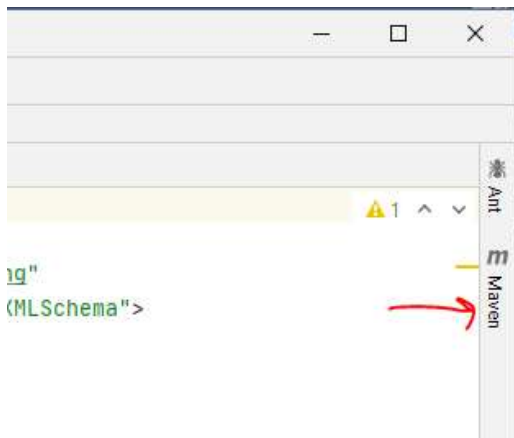
Wherever you see the text **Exercise15-Client** in the screenshots, you should see **Lab10-Client** in IntelliJ

Open the given **Lab10-Service** project, and the **Lab10-Client** project. In the Service project you see the schema file **greeting.xsd** in the folder **src/main/resources/xsds**

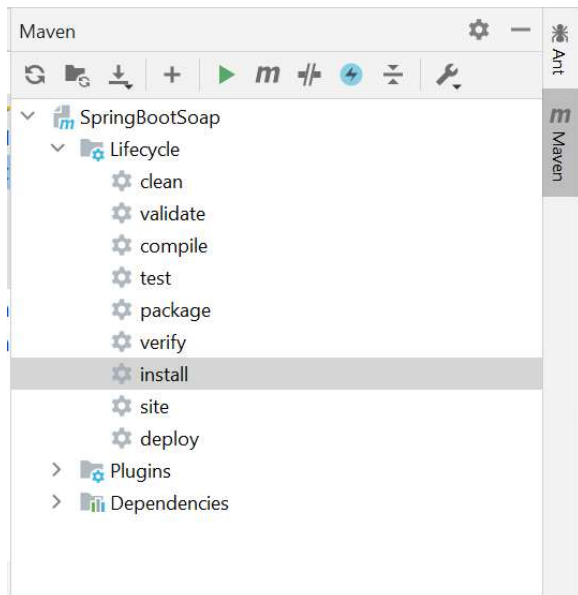


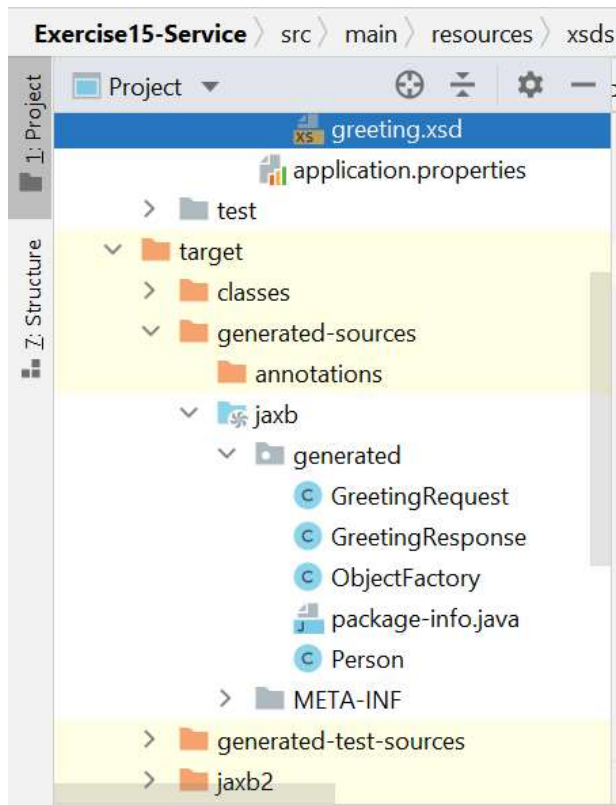
From this xsd file we need to generate the JAXB annotated Java classes

In IntelliJ, click the **Maven** tab at the right hand side of the IntelliJ window.



In the Maven window, double-click the **install** lifecycle.





Notice that this will generate the JAXB annotated classes based on the **greeting.xsd** file in the **resources/xsds** folder.

The GreetingEndpoint.java uses these generated classes:

```
@Endpoint
public class GreetingEndpoint {
    private static final String NAMESPACE_URI = "http://springtraining/greeting";

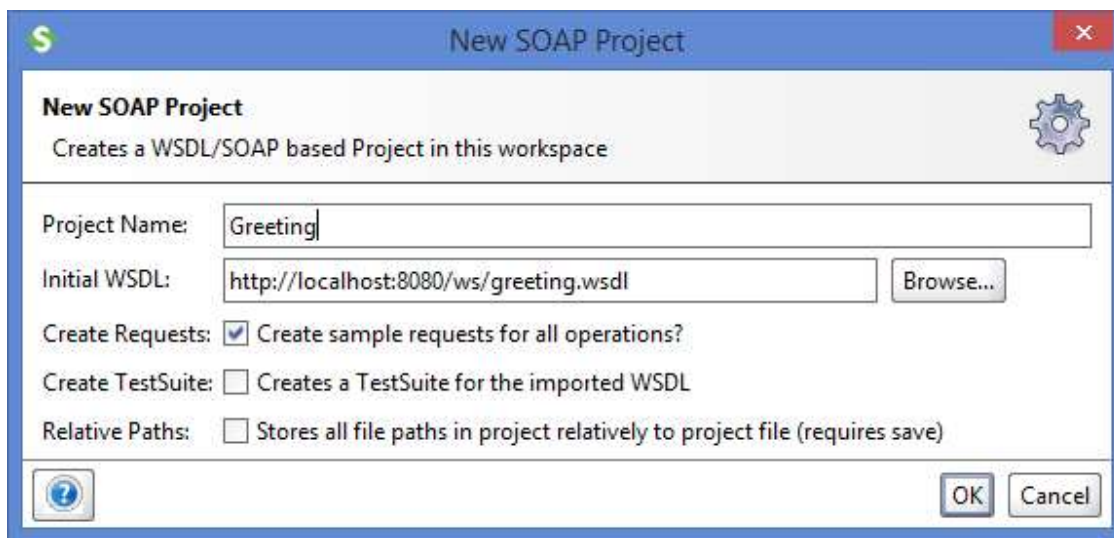
    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "GreetingRequest")
    @ResponsePayload
    public GreetingResponse getGreeting(@RequestPayload GreetingRequest request) {
        GreetingResponse response = new GreetingResponse();
        response.setGreeting(request.getPerson().getFirstName()+" "+request.getPerson().getLastName());
        return response;
    }
}
```

First run the file SpringBootSoapApplication.java. The service is now deployed to tomcat.

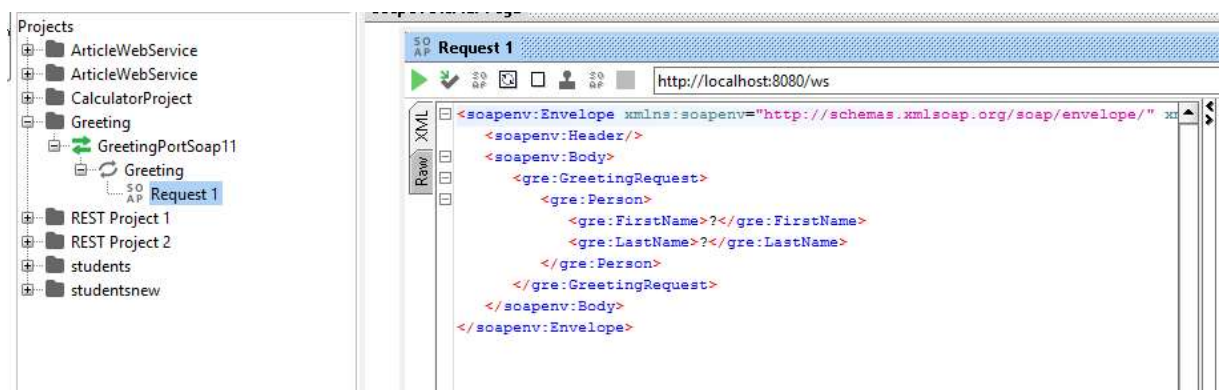
The download SoapUI from <https://www.soapui.org/downloads/soapui.html> and install SoapUI.

Then run SoapUI.

In SoapUI select **File->New SOAP Project**.



Fill in the Project Name **Greeting** and the Initial WSDL <http://localhost:8080/ws/greeting.wsdl>. Then click **OK**.



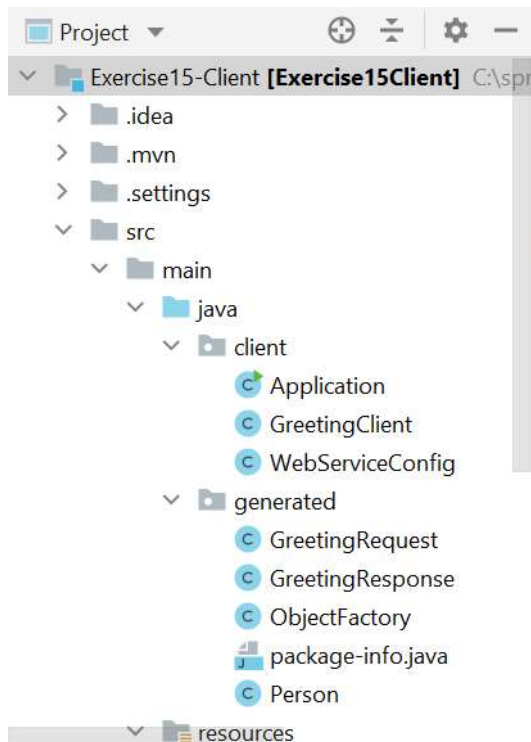
Open the Greeting project, and double click **Request 1**, and you see an example request based on the WSDL.



Fill in a name in the FirstName and LastName tag and click the Submit button which is the green triangle at the top left of the **Request 1** page. The SOAP request is now send to our service, and the response is shown in SoapUI.



Now we will look at the given **Lab10-Client** project



In this project we copied the generated classes from **Lab10-Service** . Then we implemented **GreetingClient.java** to work with these generated classes.

```
public class GreetingClient extends WebServiceGatewaySupport {  
  
    public String getMessage(Person person) {  
        GreetingRequest request = new GreetingRequest();  
        request.setPerson(person);  
  
        GreetingResponse response = (GreetingResponse)  
            getWebServiceTemplate().marshalSendAndReceive(request);  
        return response.getGreeting();  
    }  
}
```

Now run Application.java and you should see the following in the console:

Frank Brown

Now we are going to implement a SOAP calculator webservice with Spring boot.

In IntelliJ, right-click the project **Lab10-Service** and select **Copy->Copy**

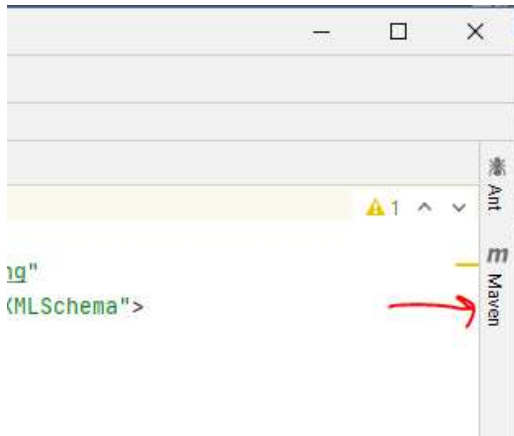
Then right-click in the same project window and select **Paste**

Give the new project the name **Lab10Calculator**

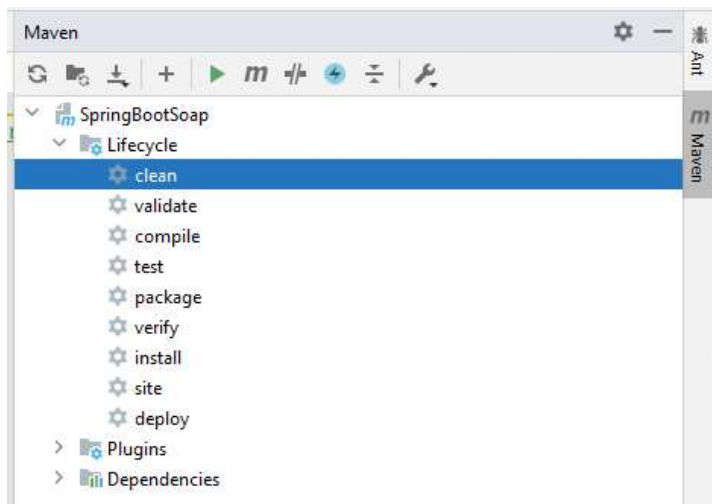
Now open the just created **Lab10Calculator** project in IntelliJ. Delete the file `src/main/resources/xsds/greeting.xsd`. Create a new file in `src/main/resources/xsds` with the name **calculator.xsd** and copy and paste the following content in this file:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="unqualified" elementFormDefault="qualified"
  targetNamespace="http://springtraining/calculator">
  <xs:element name="AddRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="number1" type="xs:int" />
        <xs:element name="number2" type="xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SubtractRequest">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="number1" type="xs:int" />
        <xs:element name="number2" type="xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="AddResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SubtractResponse">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="result" type="xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


In IntelliJ, click the **Maven** tab at the right hand side of the IntelliJ window.



In the Maven window, double-click the **clean** lifecycle.



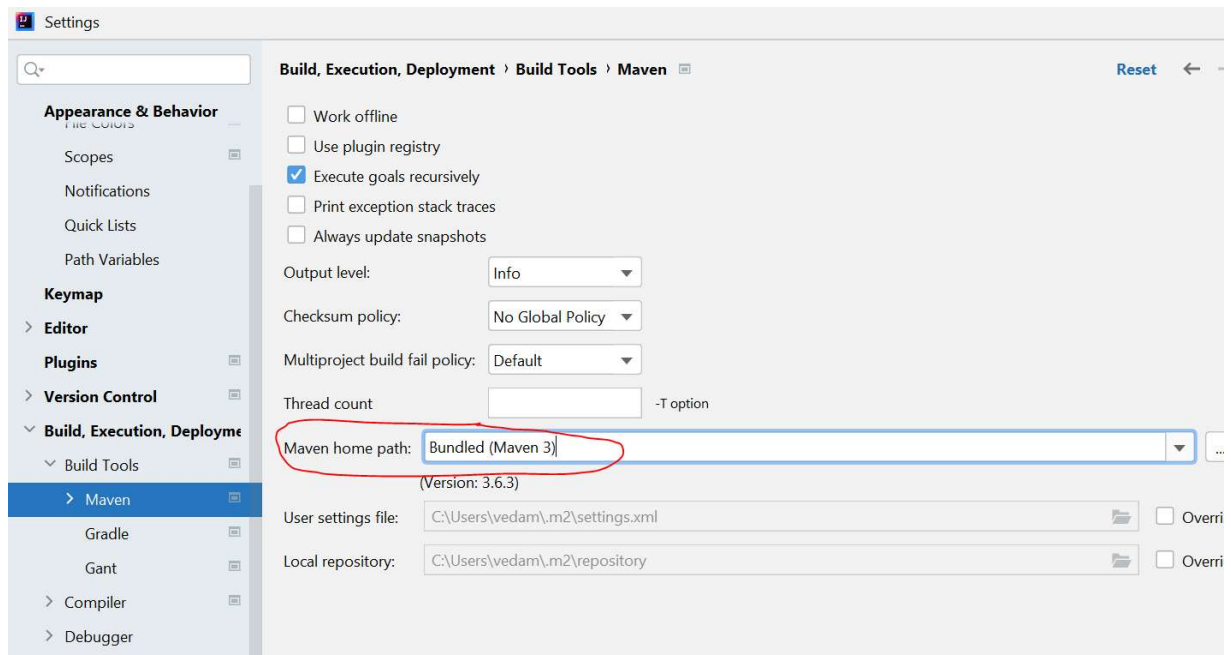
Notice that the whole **target** directory is removed.

If Maven does not work for you then do the following:

In IntelliJ, select **File -> Settings**.

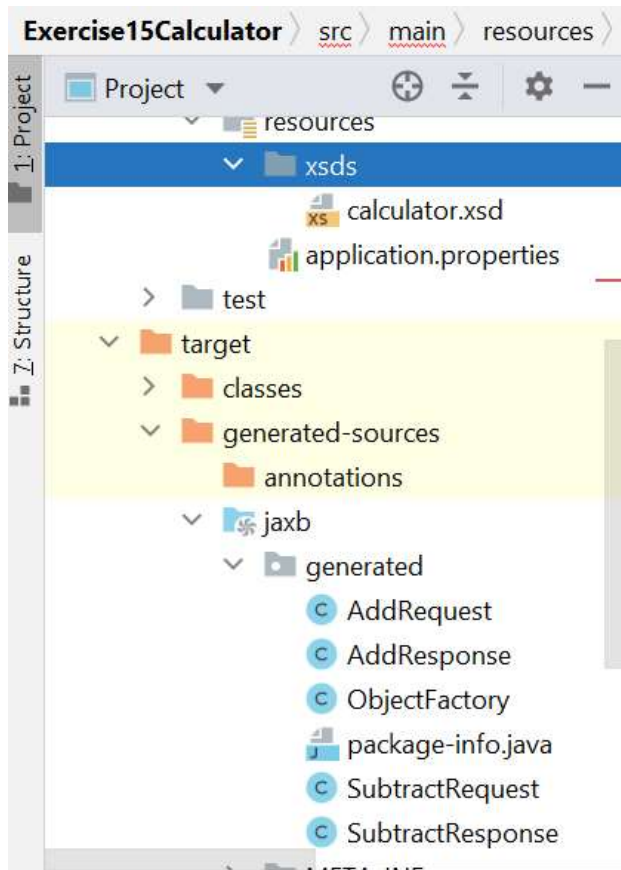
Then go to the **Build, Execution, Deployment** tab

Select **Build Tools -> Maven**



Make sure the **Bundled (Maven 3)** is selected at **Maven home path**.

Now double-click the **install** lifecycle in the maven window.



Notice that this will generate the JAXB annotated classes based on the calculator.xsd file in the **resources/xsds** folder.

Then delete the file GreetingEndpoint.java.

Then write a Calculator class as a Spring component:

```
@Component
public class Calculator {
    public int add(int x, int y){
        return x+y;
    }
    public int subtract(int x, int y){
        return x-y;
    }
}
```

Now we write the Calculator endpoint as follows:

```
import generated.AddRequest;
import generated.AddResponse;
import generated.SubtractRequest;
import generated.SubtractResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.ws.server.endpoint.annotation.Endpoint;
import org.springframework.ws.server.endpoint.annotation.PayloadRoot;
import org.springframework.ws.server.endpoint.annotation.RequestPayload;
import org.springframework.ws.server.endpoint.annotation.ResponsePayload;

@Endpoint
public class CalculatorEndpoint {

    @Autowired
    Calculator calculator;

    private static final String NAMESPACE_URI =
"http://springtraining/calculator";

    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "AddRequest")
    @ResponsePayload
    public AddResponse add(@RequestPayload AddRequest request) {
        AddResponse response = new AddResponse();
        int calcresult= calculator.add(request.getNumber1(),
request.getNumber2());
        response.setResult(calcresult);
        return response;
    }

    @PayloadRoot(namespace = NAMESPACE_URI, localPart = "SubtractRequest")
    @ResponsePayload
    public SubtractResponse add(@RequestPayload SubtractRequest request) {
        SubtractResponse response = new SubtractResponse();
        int calcresult= calculator.subtract(request.getNumber1(),
request.getNumber2());
        response.setResult(calcresult);
        return response;
    }
}
```

The last thing we have to do is to modify `WebServiceConfig` as follows:

```
@EnableWs
@Configuration
public class WebServiceConfig extends WsConfigurerAdapter {
    @Bean
    public ServletRegistrationBean messageDispatcherServlet(ApplicationContext
applicationContext) {
        MessageDispatcherServlet servlet = new MessageDispatcherServlet();
        servlet.setApplicationContext(applicationContext);
        servlet.setTransformWsdlLocations(true);
        return new ServletRegistrationBean(servlet, "/ws/*");
    }

    @Bean(name = "calc")
    public DefaultWsdll11Definition calcdefaultWsdll11Definition(XsdSchema
calcSchema) {
        DefaultWsdll11Definition wsdl11Definition = new
DefaultWsdll11Definition();
        wsdl11Definition.setPortTypeName("CalcPort");
        wsdl11Definition.setLocationUri("/ws");

        wsdl11Definition.setTargetNamespace("http://springtraining/calculator");
        wsdl11Definition.setSchema(calcSchema);
        return wsdl11Definition;
    }

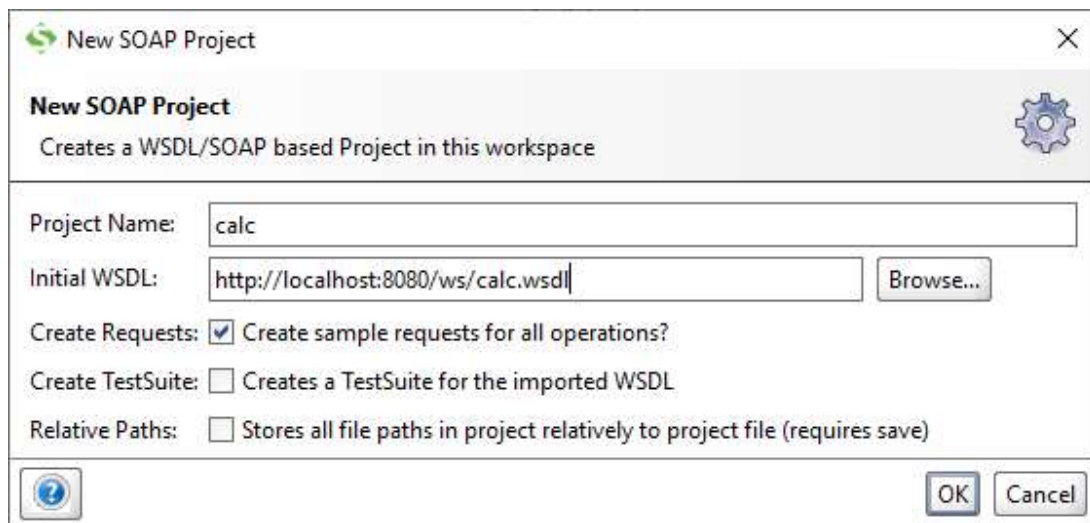
    @Bean
    public XsdSchema calcSchema() {
        return new SimpleXsdSchema(new
ClassPathResource("xsds/calculator.xsd"));
    }
}
```

Now run the application.

Check if you can get the WSDL file with the URL: <http://localhost:8080/ws/calc.wsdl>



In SoapUI create a new SOAP project with the calculator WSDL link:



The 'New SOAP Project' dialog box in SoapUI. It has a title bar with a green icon and a close button. The main area is titled 'New SOAP Project' with a subtitle 'Creates a WSDL/SOAP based Project in this workspace' and a gear icon. Below this are four fields: 'Project Name' with the value 'calc', 'Initial WSDL' with the value 'http://localhost:8080/ws/calc.wsdl' and a 'Browse...' button, 'Create Requests' with a checked checkbox and the text 'Create sample requests for all operations?', and 'Create TestSuite' with an unchecked checkbox and the text 'Creates a TestSuite for the imported WSDL'. At the bottom, there is a 'Relative Paths' section with an unchecked checkbox and the text 'Stores all file paths in project relatively to project file (requires save)', and 'OK' and 'Cancel' buttons.

New SOAP Project
Creates a WSDL/SOAP based Project in this workspace

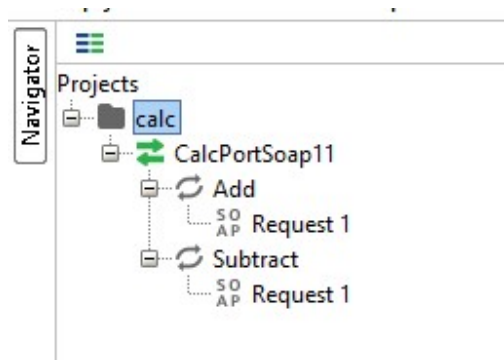
Project Name:

Initial WSDL:

Create Requests: ☒ Create sample requests for all operations?

Create TestSuite: ☐ Creates a TestSuite for the imported WSDL

Relative Paths: ☐ Stores all file paths in project relatively to project file (requires save)



Now test if the service works correctly



The SoapUI Raw XML view showing the request and response for the 'Add' operation. The left pane shows the request XML, and the right pane shows the response XML. The request is a SOAP envelope with a header and a body containing a 'cal:AddRequest' element with two numbers, 7 and 5. The response is a SOAP envelope with a header and a body containing a 'ns2:AddResponse' element with a result of 12.

Raw XML

Request:

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <cal:AddRequest>
      <cal:number1>7</cal:number1>
      <cal:number2>5</cal:number2>
    </cal:AddRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
<?xml version='1.0' encoding='UTF-8'>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:AddResponse xmlns:ns2="http://calculator.wso2.com/calculator/">
      <ns2:result>12</ns2:result>
    </ns2:AddResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Raw XML

Request:

```
<?xml version='1.0' encoding='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <cal:SubtractRequest>
      <cal:number1>10</cal:number1>
      <cal:number2>5</cal:number2>
    </cal:SubtractRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:

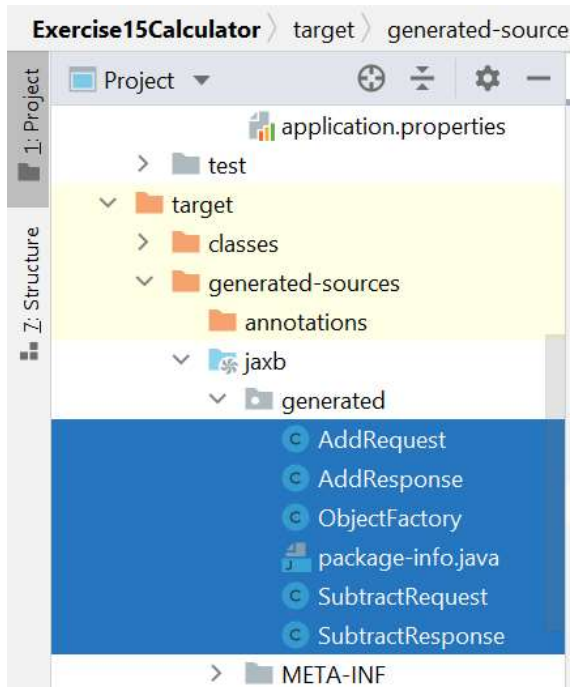
```
<?xml version='1.0' encoding='UTF-8'>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns2:SubtractResponse xmlns:ns2="http://calculator.wso2.com/calculator/">
      <ns2:result>5</ns2:result>
    </ns2:SubtractResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Modify the Calculator webservice so that we can also multiply.

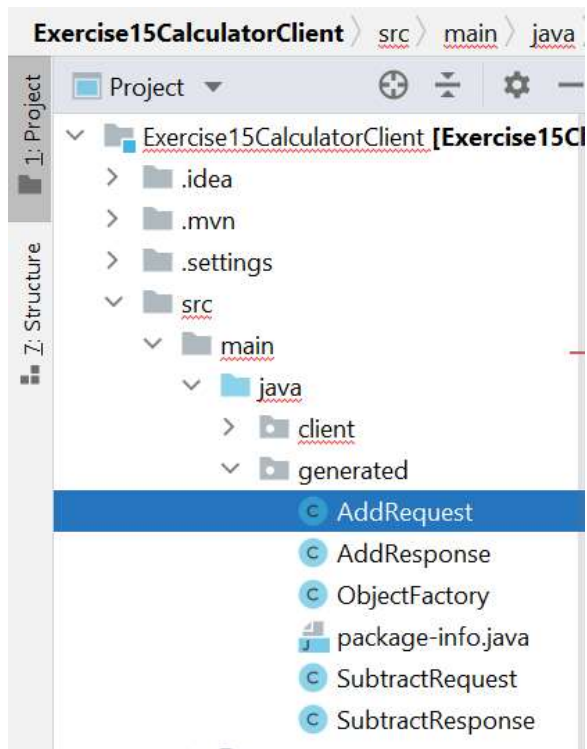
Now Copy and paste the project **Lab10-Client** to **Lab10CalculatorClient**

Open the just created project **Lab10CalculatorClient**

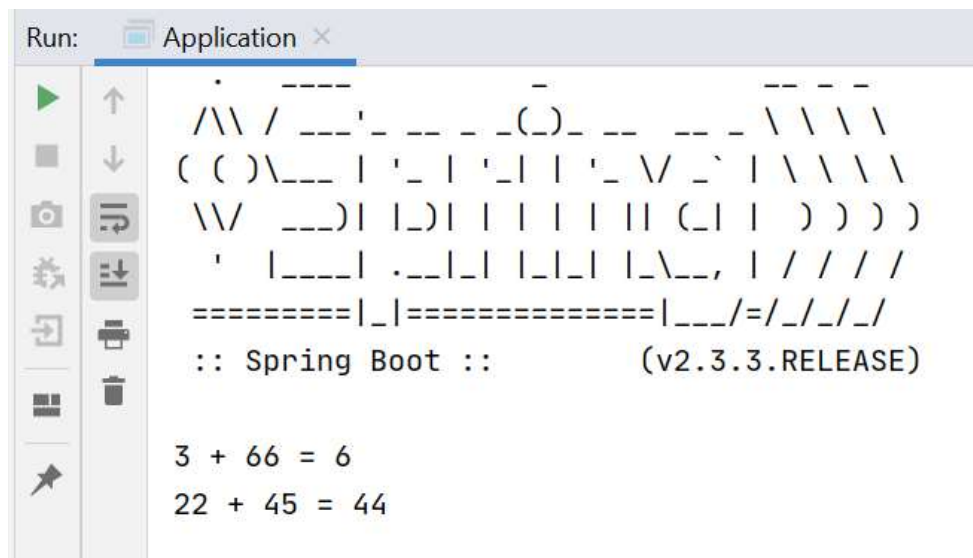
Then copy the files from **target/generated-sources/jaxb/generated** from the project **Lab10Calculator**



And paste these files in **src\main\java\generated** in the project **Lab10CalculatorClient**



Modify the client project so that we can call the calculator webservice.



What to hand in:

1. A separate zip file with the solution of this lab