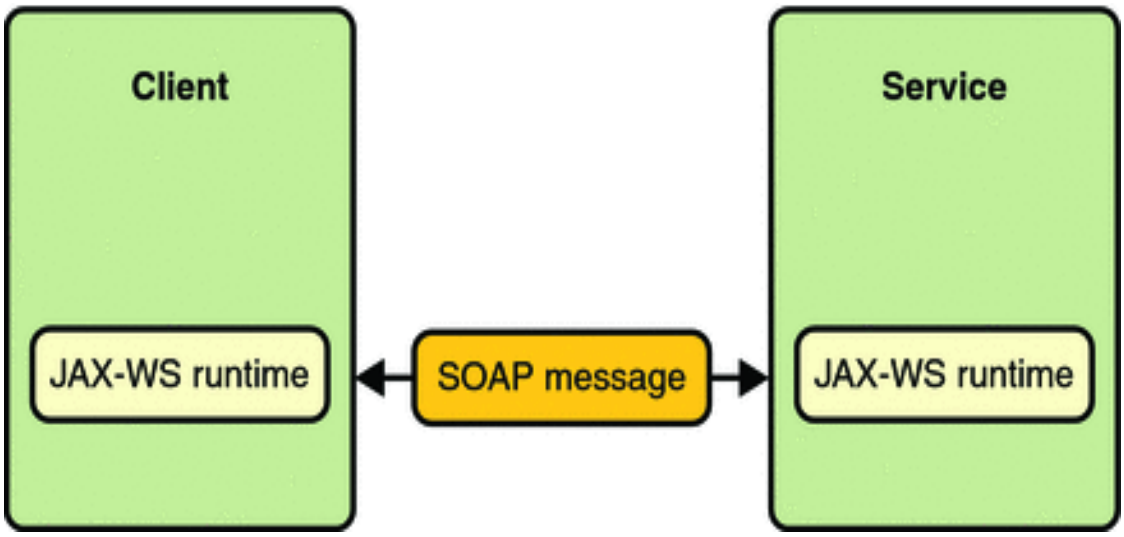
JAVA WEB SERVICES

Chapter 01 - JAX-WS

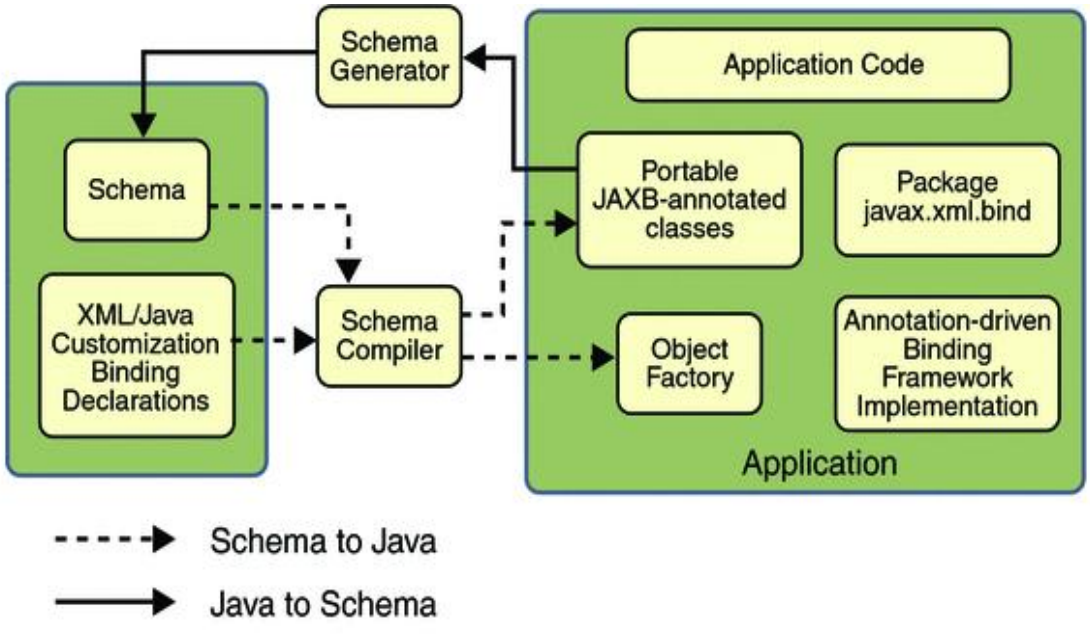
EINFÜHRUNG

- Die Java API für XML Web Services (JAX-WS) wird für die Erstellung von SOAP-Web Services verwendet
- JAX-WS ist eine der Java XML APIs
- Ist ein Part der Java EE Platform
- Verbindet eine „*web service endpoint interface*“ mit einem Java Interface
- Delegiert das Mapping von Java Daten Typen von und nach XML

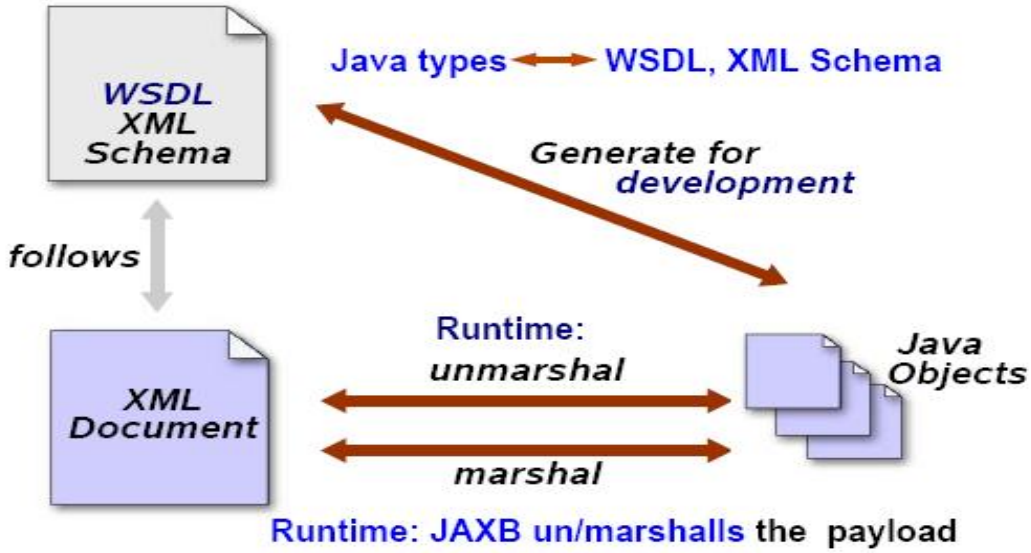
EINFÜHRUNG



JAXB



JAX-WS ⇔ JAXB



CODE-FIRST

1. Implementiere die „*web service class*“
2. Paketiere den Code in ein „*web archive*“
3. Deploye das „*web archive*“ in den „*web container*“

CONTRACT-FIRST

1. erstelle oder „bekomme“ das WSDL Dokument
2. generiere das Java Web Service Interface mittels wsimport
3. implementiere das generierte Java Web Service Interface
4. annotiere die implementierte Klasse mit @WebService
5. kompiliere die Web Service Implementation
6. paketiere die Klassen in ein „*web archive*“ und füge die wsdl zu *WEB-INF/wsdl* hinzu
7. deploye das „*web archive*“

DEPLOYMENT DESCRIPTOR

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi=„http://www.w3……>
  <servlet>
    <servlet-name>Customer</servlet-name>
    <servlet-class>org.example.service.CustomerSoap</servlet-
class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Customer</servlet-name>
    <url-pattern>/soap/customer</url-pattern>
  </servlet-mapping>
</web-app>
```

SERVICE CLASS

- Services müssen mit @WebServices annotiert werden.
- Dabei werden die Namen des Web Services definiert

```
@WebService(name = "Customer")
public class CustomerSoap {
……
```


SERVICE METHODS

- Methoden müssen JAXB kompatible Parameter und Rückgabewerte haben
- Die *@WebMethod*, *@WebParam* und *@WebResult* Anntationen können zum Anpassen der Namen, Parametern und Rückgabewerte verwendet werden

```
@WebMethod(operationName = "list")
@WebResult(name = "customer")
public List<Customer> list(@WebParam(name = "max") int max) {
   return customerBean.getAllCustomers(max);
}
```

EXCEPTION

- *JAX-WS exceptions* beinhalten immer Fehlerinformationen zusätzlich zu der Fehlermeldung
- *@WebFault annotation* können verwendet werden um den Namen der Fehlerinformation zu definieren

```
@WebFault(name = "name")
public class NotFoundException extends Exception {

……

   public String getFaultInfo() {
      return name;
   }
```


JAVA WEB SERVICES

Chapter 02 - JAX-WS

GENERATION OF CLIENT ARTIFACTS

- wsimport wird verwendet um Client Artefakte aus einem WSDL Dokument zu generieren
- wsimport [options] <WSDL_URI>
- [options]
  - -Xnocompile
    - Java Dateien werden nicht kompiliert
  - -b <path>
    - definiert *jaxws/jaxb binding* oder zusätzliche Schemas
  - -d <directory>
    - definiert das Daten Output Verzeichnis für die generierten Dateien

CLIENT

- Der Client kann mittels einer *service factory* einen *stub* erstellen auf welchem die Methoden aufgerufen werden

```
public static void main(String[] args) {
   CustomerBaseService service = new CustomerBaseService();
   customerbase = service.getCustomerBase();
   List<Customer> customers = customerbase.list()
   …
}
```

@WEBSERVICE

@WebService(……)

- name
  - definiert den Namen des Web Services und wird als portType Namen im WSDL verwendet
  - default: Java Klassennamen
- targetNamespace
  - definiert den XML namespace für das generierte WSDL und die XML Elemente
  - default: Paketname

@WEBSERVICE

@WebService(……)

- wsdlLocation
  - definiert die URL des verwendeten WSDL Dokuments. Dieses Attribut wird nur verwendet wenn ein bereits vorhandenes WSDL verwendet wird.
- serviceName
  - definiert den Service Namen.
- portName
  - definiert den WSDL Port welcher verwendet werden soll.

ENDPOINT ADDRESS

- Servlet-Based web service

  http://<host>:<port>/<context-root>/<servlet-url-mapping>

- EJB-Based web service

  http://<host>:<port>/<service-name>/<port-name>

sirlordrobinson@sirlordrobinson-Lenovo-Yoga-2-Pro:~$ sudo su
[sudo] Passwort für sirlordrobinson:

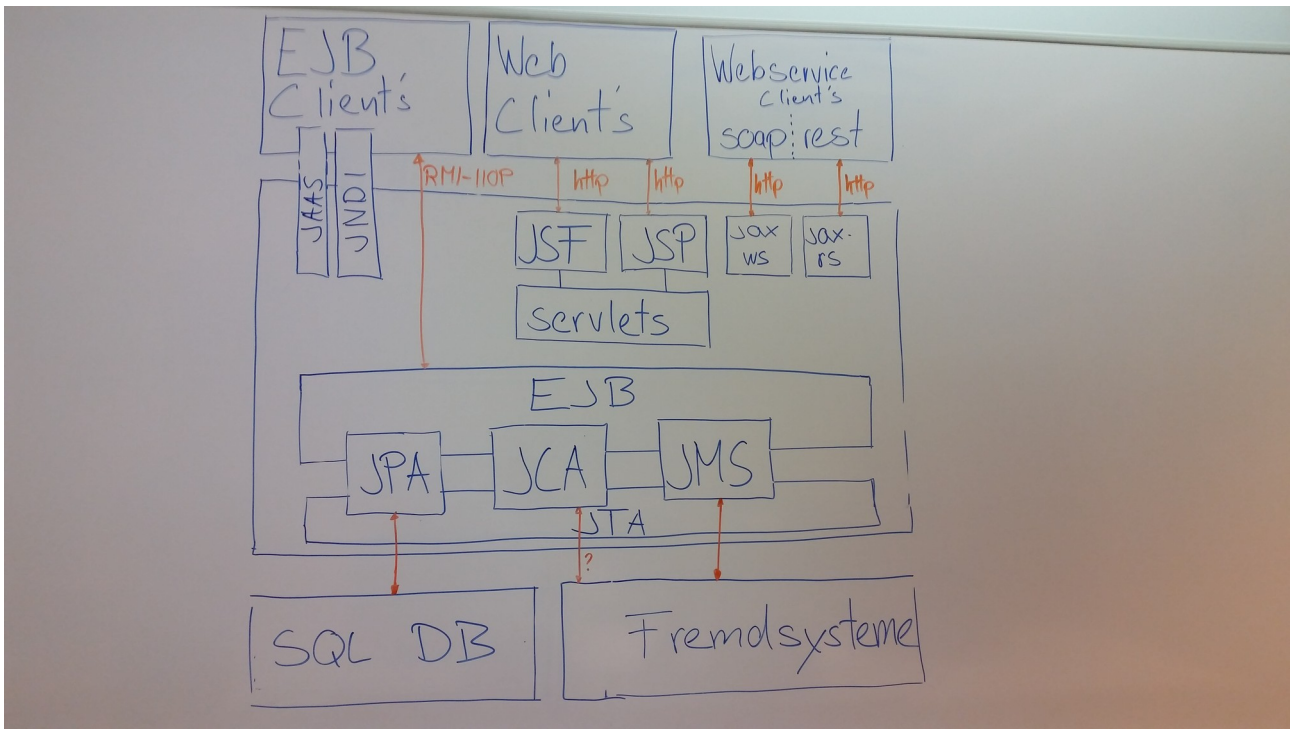
root@sirlordrobinson-Lenovo-Yoga-2-Pro:/opt/SoapUI-5.2.1# bin/soapui.sh

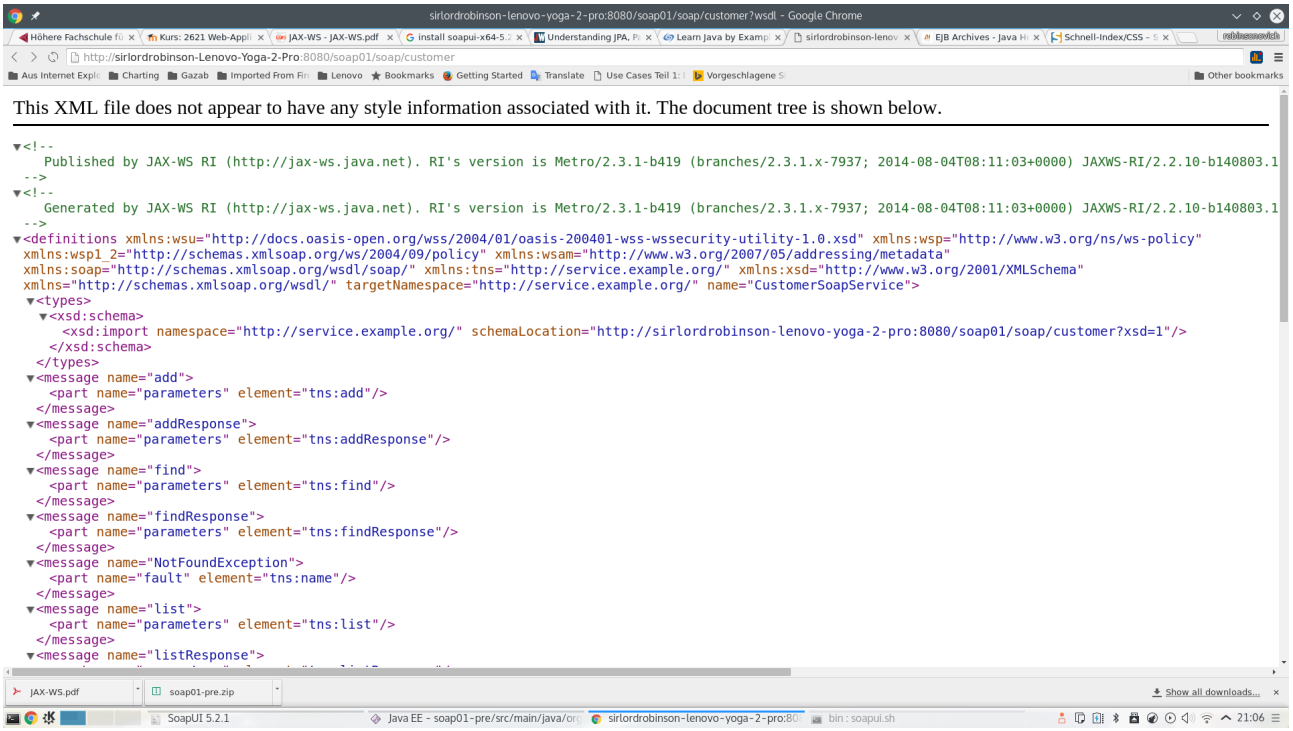
1. Deploy projekt on webserver eg. Glassfish
   then copy the below code from Glassfish console.

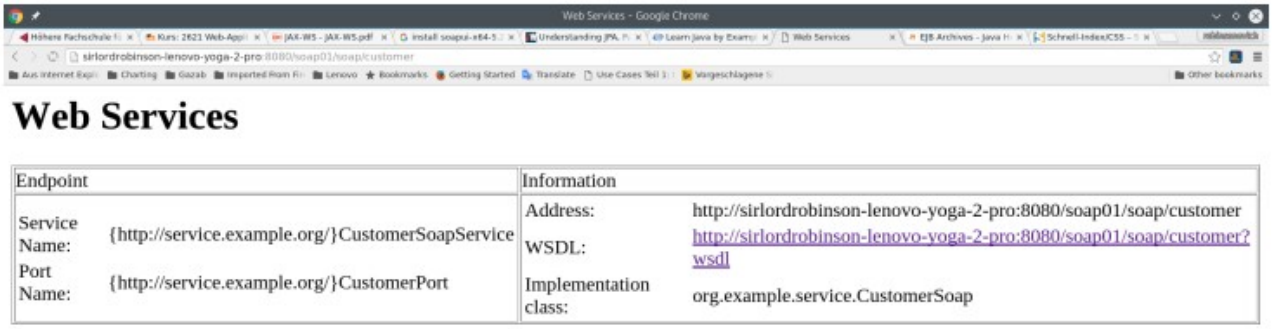
2. http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01_pre/soap/customer
3. Paste the above address into the browser, then code on the second line will be generated automatically.

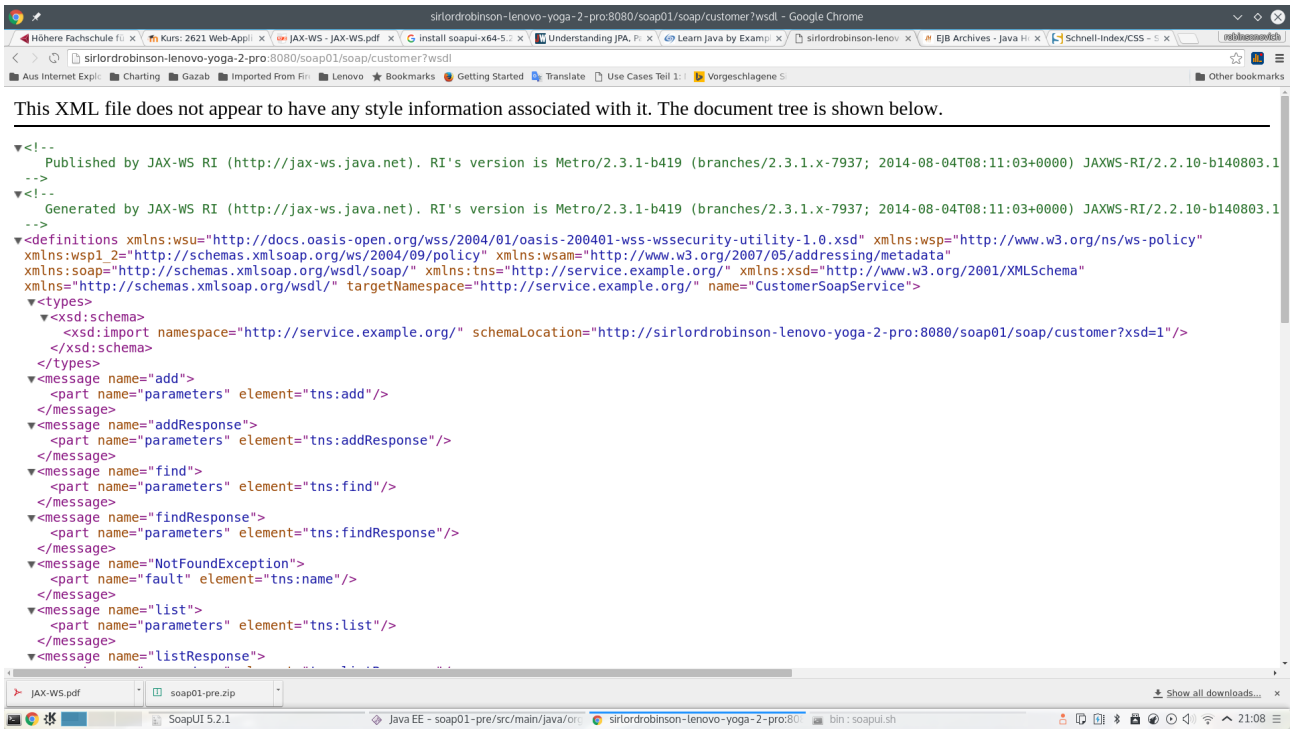
4. Paste below code come into the soapUI

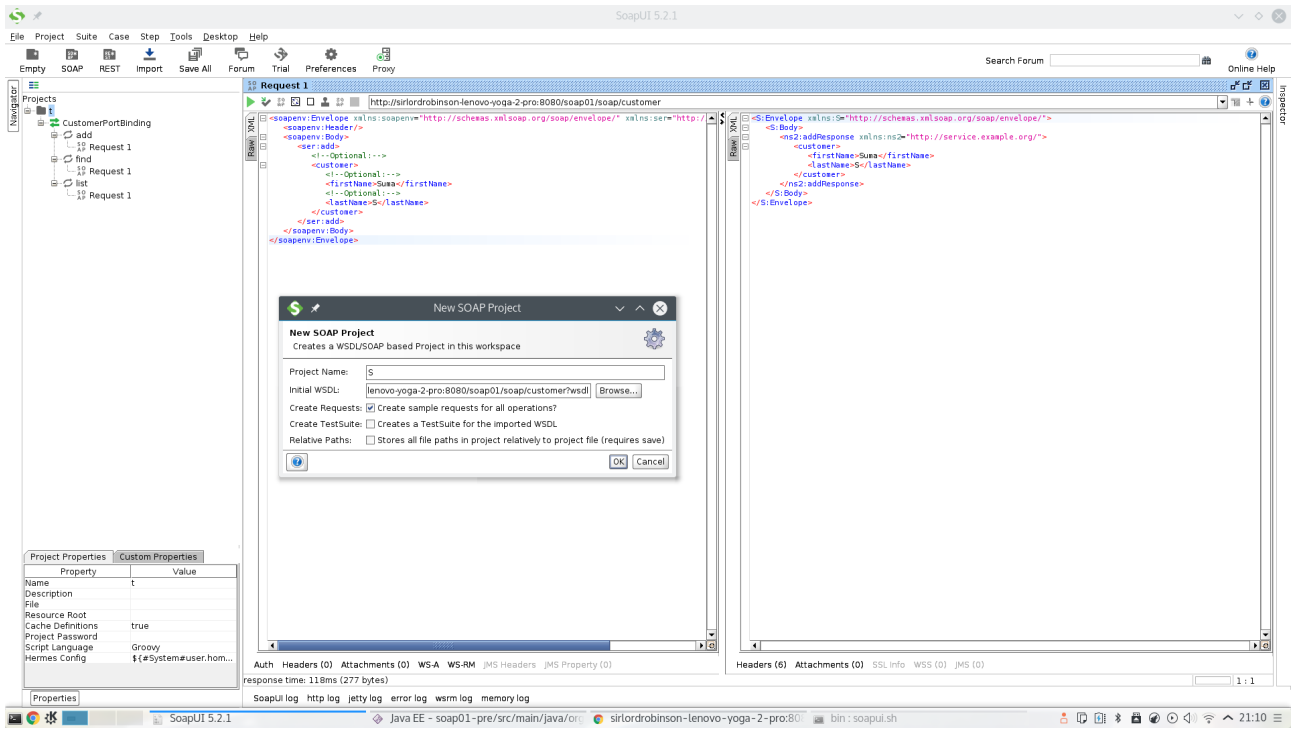
4. http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01_pre/soap/customer?wsdl


EJB Client's
Web Client's
Webservice Client's
soap rest
JAAS
JNDI
RMI-IIOP
http
http
http
http
JSF
JSP
jax ws
jax rs
servlets
EJB
JPA
JCA
JMS
JTA
?
SQL DB
Fremdsysteme


sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?wsdl - Google Chrome
http://sirlordrobinson-Lenovo-Yoga-2-Pro:8080/soap01/soap/customer
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<!--
Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) JAXWS-RI/2.2.10-b140803.1
-->
<!--
Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) JAXWS-RI/2.2.10-b140803.1
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service.example.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://service.example.org/" name="CustomerSoapService">
<types>
<xsd:schema>
<xsd:import namespace="http://service.example.org/" schemaLocation="http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?xsd=1"/>
</xsd:schema>
</types>
<message name="add">
<part name="parameters" element="tns:add"/>
</message>
<message name="addResponse">
<part name="parameters" element="tns:addResponse"/>
</message>
<message name="find">
<part name="parameters" element="tns:find"/>
</message>
<message name="findResponse">
<part name="parameters" element="tns:findResponse"/>
</message>
<message name="NotFoundException">
<part name="fault" element="tns:name"/>
</message>
<message name="list">
<part name="parameters" element="tns:list"/>
</message>
<message name="listResponse">
JAX-WS.pdf
soap01-pre.zip
Show all downloads...
SoapUI 5.2.1
21:06


Web Services - Google Chrome
sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer
Web Services
Endpoint
Service Name: {http://service.example.org/}CustomerSoapService
Port Name: {http://service.example.org/}CustomerPort
Information
Address: http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer
WSDL: http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?wsdl
Implementation class: org.example.service.CustomerSoap


sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?wsdl - Google Chrome
sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?wsdl
This XML file does not appear to have any style information associated with it. The document tree is shown below.
<!--
Published by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) JAXWS-RI/2.2.10-b140803.1
-->
<!--
Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is Metro/2.3.1-b419 (branches/2.3.1.x-7937; 2014-08-04T08:11:03+0000) JAXWS-RI/2.2.10-b140803.1
-->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://service.example.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://service.example.org/" name="CustomerSoapService">
<types>
<xsd:schema>
<xsd:import namespace="http://service.example.org/" schemaLocation="http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer?xsd=1"/>
</xsd:schema>
</types>
<message name="add">
<part name="parameters" element="tns:add"/>
</message>
<message name="addResponse">
<part name="parameters" element="tns:addResponse"/>
</message>
<message name="find">
<part name="parameters" element="tns:find"/>
</message>
<message name="findResponse">
<part name="parameters" element="tns:findResponse"/>
</message>
<message name="NotFoundException">
<part name="fault" element="tns:name"/>
</message>
<message name="list">
<part name="parameters" element="tns:list"/>
</message>
<message name="listResponse">
JAX-WS.pdf
soap01-pre.zip
Show all downloads...
SoapUI 5.2.1
Java EE - soap01-pre/src/main/java/or
sirlordrobinson-lenovo-yoga-2-pro:80
bin : soapui.sh
21:08


SoapUI 5.2.1
File Project Suite Case Step Tools Desktop Help
Empty SOAP REST Import Save All Forum Trial Preferences Proxy
Search Forum
Online Help
Projects
CustomerPortBinding
add
Request 1
find
Request 1
list
Request 1
Request 1
http://sirlordrobinson-lenovo-yoga-2-pro:8080/soap01/soap/customer
New SOAP Project
New SOAP Project
Creates a WSDL/SOAP based Project in this workspace
Project Name: S
Initial WSDL: lenovo-yoga-2-pro:8080/soap01/soap/customer?wsdl
Browse...
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)
OK
Cancel
Project Properties
Custom Properties
Property
Value
Name
Description
File
Resource Root
Cache Definitions
Project Password
Script Language
Hermes Config
true
Groovy
${#System#user.hom...
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0)
response time: 118ms (277 bytes)
Headers (6) Attachments (0) SSL Info WSS (0) JMS (0)
SoapUI log http log jetty log error log wsrm log memory log
SoapUI 5.2.1
Java EE - soap01-pre/src/main/java/or
sirlordrobinson-lenovo-yoga-2-pro:80
bin : soapui.sh
21:10

To Deploy a project on Glassfish Server in Eclipse, do

1. Right click on the project, choose run As, Run on server, glassfish server.

SoupUI installation in Linux

http://askubuntu.com/questions/113159/is-there-a-standalone-version-of-soapui-for-linux

Then to webbrowser see link in image 1 above.

SOAP Exercise 01: Introduction

- SoapUI installieren
- Source Project soap01 importieren
- DeploymentDescriptor anpassen (uri: soap/customer)
- Service Klasse CustomerSoap erstellen (org.example.service)
- Methode list() implementieren
- Methode find() (mittels firstName, lastName) implementieren
- Methode add() implementieren
- Application deployen und mit SoapUI testen

SOAP Exercise 02: Client / wsimport

- soap01 service auf glassfish deployen
- soap01-client in eclipse workspace entpacken
- mittels wsimport die client-artifacts erstellen (endpoint-address ?wsdl)
- client-artifacts überprüfen
- soap01-client ausprogrammieren

SOAP Exercise 03: externer SOAP Service einsetzen

Aufgabe

- Erweitere die Klasse SoapClient:
- Der Kommandozeilen-Client SoapClient (*ch.hftm.webservices.soap.client*) kann die Temperatur von Celsius in Fahrenheit umrechnen und umgekehrt.
- Das Programm ist bereits implementiert, es muss lediglich der SOAP Service angebunden werden. Und die entsprechenden Methoden implementiert werden.
- Zeile: 14, 50, 63 (exclusive import, deklarationen, etc)
- Service:http://www.w3schools.com/webservices/tempconvert.asmx?WSDL
- Folgende Ausgabe soll erzielt werden:

```
1 convert from °C to F
2 convert from F to °C
0 Exit
> 1
Celsius:  28
28 °C entsprechen 82.4 Fahrenheit
```

bzw.

```
1 convert from °C to F
2 convert from F to °C
0 Exit
> 2
Fahrenheit:  82.4
82.4 Fahrenheit entsprechen 28 °C
```

JAVA WEB SERVICES

Chapter 01 - REST

EINFÜHRUNG

- REST steht für **Re**presentational **S**tate **T**ransfer
- REST ist eine Architektur Pattern für die Entwicklung von Web Services
- REST Web Services kommunizieren mittels HTTP, dabei wird auch das HTTP „Vokabular“ verwendet
  - Methoden (GET, POST, etc.)
  - HTTP URI syntax (paths, parameters, etc.)
  - Media types (xml, json, html, plain text, etc)
  - HTTP Antwort Codes

EINFÜHRUNG

- Representational
  - Der Client verarbeitet die Informationen welche für die Identifizierung, Änderung oder Löschung notwendig sind
- State
  - Informationen der Web Resource werden auf dem Client gespeichert
- Transfer
  - Client Status wird mittels HTTP ausgetauscht

EINFÜHRUNG

- Die sechs Charakteristiken von REST
  - Uniform interface
  - Decoupled client-server interaction
  - Stateless
  - Cacheable
  - Layered
  - Extensible through code on demand (optional)

REQUEST BASICS

- Der **HTTP request** (von dem Client gesendet)
  - Identifiziert den „Ort“ der **Resource**
  - Spezifiziert die HTTP **Methode** welche beim Zugriff auf die Resource ausgeführt wird
  - Liefert optionale **Request Headers** (name-value Paare) welche zusätzliche Informationen bereitstellen (für die Verarbeitung des Requests)
  - Liefert ein optionaler **Request Body** welcher zusätzliche Daten (z.B. Form Parameters, Anhänge, etc.) dem Service bereitstellt

REQUEST BASICS

Beispiel Client Request:

Ein typischer GET Request:

```
GET /view?id=1 HTTP/1.1
User-Agent: Chrome
Accept: application/json
[CRLF]
```

Ein typischer POST Request:

```
POST /save HTTP/1.1
User-Agent: IE
Content-Type: application/x-www-form-urlencoded
[CRLF]
name=x&id=2
```

RESPONSE BASICS

- Die **HTTP Response** (von dem Service versendet)
  - Liefert den **Status** des verarbeiteten Requests
  - Liefert **Response Headers** (name-value Paare) welche zusätzliche Informationen über die Antwort liefern
  - Liefert ein optionaler **Response Body** welcher zusätzliche Daten (html, xml, binary data, etc.) beinhaltet

RESPONSE BASICS

Beispiel Server Response:

Eine typische Server Response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1337
[CRLF]
<html>
  <!-- Some HTML Content. -->
</html>
```

Ein typischer Server Error:

```
HTTP/1.1 500 Internal Server Error
```

METHODS BASICS

- HTTP Methods welche von REST unterstützt werden:
  - GET – Anfrage einer Resource an der angegebenen URL
  - POST – Übermittelt dem Service Informationen
  - PUT – Aktualisierung einer Resource
  - DELETE – Löschen einer Resource

HTTP URL BASICS

- Das **protocol** identifiziert das „Transport Schema“ welches für den Request und die Response verwendet wird
- Der **host name** identifiziert die Server Adresse
- Der **path** und **query string** wird für die Identifizierung und den Zugriff der Resource verwendet

ANNOTATIONS

- JAX-RS Annotations werden für die Konfiguration des REST Services verwendet:
  - Identifizieren der HTTP Methode für den Zugriff einer Resource
  - Festlegen eines relativen Pfades für den Zugriff einer Resource
  - Festlegen wie *query* und *form parameters*, *headers*, *cookies* und andere Teile der *HTTP request message* mit Java Methoden und Attributen übereinstimmen
  - Festlegen der verfügbaren *content types* welche Konsumiert (consumed) oder Produziert (produced) werden

METHOD ANNOTATIONS

- JAX-RS HTTP Method Annotations:
  - @GET @POST @PUT @DELETE
- Anwendung auf Java Methoden zur Bindung an eine HTTP Methode
- Lediglich eine HTTP Annotation kann auf eine Java Methode angewendet werden
- Mehreren Java Methoden können identische HTTP Methoden zugewiesen werden, solange sie unter einem anderen Pfad veröffentlicht sind

PATH ANNOTATIONS

- @Path Annotationen können verwendet werden um die Request URI anzupassen
- @Path auf einer Klassen definiert den Basispfad aller Ressourcen dieser Klassen
- @Path auf Methoden definiert den relativen Pfad zu dieser Resource
- @Path auf Methoden ist relativ zu @Path auf der Klasse
- ohne @Path auf der Klasse wird die Resource im root Verzeichnis veröffentlicht
- Da die Pfade immer relativ sind, ist ein führender forward slash (/) bei der Definition nicht notwendig

PATH ANNOTATIONS

- Die @Path Annotation unterstützt die Verwendung von *template parameters* in der Form von:

  {name : regex} ➔ @Path(,,{year:[0-9]{4}}")

- Der *template parameter name* ist dabei zwingend notwendig
- Der Doppelpunkt (:) gefolgt einer *regular expression* ist optional
- Mehrere *template parameters* können in einem @Path definiert werden ➔ @Path("{xxx}/{yyy}/{zzz}")
- *Template parameter values* werden mittels PathParam ausgelesen ➔ @PathParam ("year") int year

PARAMETER ANNOTATIONS

- Häufige JAX-RS Parameter Annotations:
  - @PathParam ➔ path segment
  - @QueryParam ➔ query string parameter
  - @FormParam ➔ Form POST parameter
  - @HeaderParam ➔ key-value pair

- In der Regel auf Eingabe Parameter innerhalb von Service Methoden angewendet
- Erweiterte Parameter Anntoationen können aus der JAX-RS API entnommen werden

PATH-PARAM (MULTIPLE)

```
@GET
@Path("{year}/{month}/{day}")
public Response getUserHistory(
      @PathParam("year") int year,
      @PathParam("month") int month,
      @PathParam("day") int day) {

      String date = year + "/" + month + "/" + day;

      return Response.status(200)
            .entity("getUserHistory is called, year/month/day : " + date)
            .build();
}
```

URI Pattern:

…../2016/02/23

QUERY-PARAM

CODE:

```
@GET
@Path("/query")
public Customer getCustomer(
      @QueryParam("from") int from,
      @QueryParam("to") int to,
      @QueryParam("orderBy") List<String> orderBy) {

   return ....;

}
```

URI Pattern:

…../query?from=100&to=200&orderBy=age&orderBy=name”

FORM-PARAM

CODE:

```
@POST
@Path("/add")
public Customer addCustomer(
       @FormParam("name") String name,
       @FormParam("age") int age) {

   return ....;

}
```

HTML:

```
<form action=„…/…/add“ method="post">
    <p>
        Name : <input type="text" name="name" />
    </p>
    <p>
        Age : <input type="text" name="age" />
    </p>
    <input type="submit" value="Add User" />
</form>
```

HEADER-PARAM

CODE:

```
@GET
@Path("getCustomer")
@Produces("application/xml")
public Customer getCustomer(@HeaderParam("index") int index) {
   return customerDB.getCustomer(index);
}
```

URI Pattern:

…../getCustomer

Request Pattern:

```
GET /getCustomer HTTP/1.1
User-Agent: Chrome
Accept: application/xml
index: 1
```

MIME ANNOTATIONS

@Produces

- Definiert die Ausgabe Formate der Klasse oder Methode

@Consumes

- Definiert die Akzeptierten Eingabe Formate der Klasse oder Methode

*Methoden Anntotationen überschreiben Klassen Annotationen

HEADERS

HEADER FIELD NAME	Description	Example
Accept	Content-Types that are acceptable for the response	Accept: text/plain
Accept-Charset	Character sets that are acceptable	Accept-Charset: utf-8
Accept-Encoding	List of acceptable human languages for response	Accept-Language: en-CA

JAX-RS ANNOTATIONS

@POST, @GET, @PUT, @DELETE @HEAD, @OPTIONS	indication the HTTP request type
@Path	relative URI to the resource
@PathParam @QueryParam @FormParam @HeaderParam @CookieParam	extract HTTP parameters in the Java Class
@Consumes @Produces	MIME media types
@Provider	JAX-RS runtime informations
@Context	Request related informations

SUCCESSFUL CLIENT REQUESTS CODE

Code	Description
200	OK. The request has successfully executed. Response depends upon the verb invoked.
201	Created. The request has successfully executed and a new resource has been created in the process. The response body is either empty or contains a representation revealing URIs for the resource created. The Location header in the response should point to the new URI as well.
202	Accepted. The request was valid and has been accepted but has not yet been processed. The response should include a URI to poll for status updates on the request. This allows asynchronous REST requests.
204	No Content. The request was successfully processed but the server did not have any response. The client should not update its display.

REDIRECTED CLIENT REQUESTS CODE

Code	Description
301	Moved Permanently. The requested resource is no longer located at the specified URL. The new Location should be returned in the response header. Only GET or HEAD requests should redirect to the new location. The client should update its bookmark if possible.
302	Found. The requested resource has temporarily been found somewhere else. The temporary Location should be returned in the response header. Only GET or HEAD requests should redirect to the new location. The client need not update its bookmark as the resource may return to this URL.
303	See Other. This response code has been reinterpreted by the W3C Technical Architecture Group (TAG) as a way of responding to a valid request for a non-network addressable resource. This is an important concept in the Semantic Web when we give URIs to people, concepts, organizations, etc. There is a distinction between resources that can be found on the Web and those that cannot. Clients can tell this difference if the get a 303 instead of 200. The redirected Location will be reflected in the Location header in the response. It will contain a reference to a document about the resource or perhaps some metadata about it. This is not a universally popular decision but is currently the provided guidance.

INVALID CLIENT REQUESTS CODE

Code	Description
400	Bad Request. Generally the sign of a malformed or otherwise invalid request.
401	Unauthorized. Without further authorization credentials, the client is not allowed to issue the request. The inclusion of an "Authorization" header with valid credentials might still succeed.
403	Forbidden. The server is disallowing the request. Extra credentials will not help.
404	Not Found. The server could not match the request to a known resource.
405	Method Not Allowed. The requested method (verb) is not allowed for that resource. Response will indicate in an "Allow" header what is allowed.
406	Not Acceptable. The server cannot generate a representation compatible with what was asked for in the request "Accept" header.
410	Gone. The resource is explicitly no longer available and will not be in the future.
411	Length Required. The server requires the client to specify a "Content- Length" header indicating the size of the request. A resubmit with this header might succeed.
413	Entity Too Large. The request entity is too large for the server to process.
415	Unsupported Media Type. The client submitted a media type that is incompatible for the specified resource.

REST Exercise 01: Introduction

Übung 1.1

- RESTClient installieren
- Demo Project rest01 importieren
- SLSB CustomerService erstellen (ohne interface)
- Pfad-Annotation "customers" erstellen (auf Klasse)
- Methode public String Hallo() erstellen (return Value "Hallo")
- rest Parameter: get / Pfad: test / produziert: xml
- Methode Testen

Übung 1.2

- Methode public List<Customer> getAllCustomers() erstellen
- rest Parameter: get / produziert: json
- Methode Testen

Übung 1.3

- Methode public Customer getCustomerPP() erstellen
- rest Parameter: get / Pfad: getCustomerPP / Pfad Parameter: index / produziert: xml
- Methode Testen

Übung 1.4

- Methode public Customer getCustomerHP() erstellen
- rest Parameter: get / Pfad: getCustomerHP / Header Parameter: index / produziert: xml
- Methode Testen

Übung 1.6

- Methode public Customer getCustomerQP() erstellen
- rest Parameter: get / Pfad: getCustomerQP / Query Parameter: index / produziert: xml
- Methode Testen

Übung 1.7

- Methode public Customer addCustomer() erstellen
- rest Parameter: post / Pfad: addCustomer / Konsumiert xml
- Methode Testen

JAVA WEB SERVICES

Chapter 02 - REST

RESPONSE OBJECT

- Mit der Rückgabe eines Response Objekt kann dem Client nebst dem Rückgabeobjekt auch ein HTTP Statuscode übermittelt werden

```
@POST
@Consumes("application/xml")
public Response createStudent(Student student) {
   schoolDB.addStudent(student);
   return Response.status(201)
         .entity(student)
         .build();
}
```

EXCEPTION HANDLING

- Mittels einer WebApplicationException kann eine HTTP Fehlermeldung erzeugt werden, welche dem Client gesendet wird

```
public class BadRequestException extends WebApplicationException {

   public BadRequestException() {
      super(Status.BAD_REQUEST);
   }

   public BadRequestException(String message) {
      super(Response.status(Status.BAD_REQUEST).entity(message).build());
   }
}
```


MESSAGE BODY READERS AND WRITERS

- *Message body readers* und *writers* bieten die Möglichkeit die representation einer Resource mit den dazugehörigen Java Typen zu verbinden
- JAX-RS Implementation unterstützt standardmässig bereits einige Konversionen nach unterschiedlichen Repräsentationen
- Alle Klassen sind als MessageBodyWriters und Readers implementiert
    - byte[]
    - java.lang.String
    - javax.xml.bind.JAXBElement
    - …
- JAXB wird für das erstellen von XML representations verwendet

MESSAGE BODY WRITER

- Um einen MessageBodyWriter zu erstellen müssen drei Methoden implementieren werden
  - **writeTo()**
    - Diese Methode wird aufgerufen sobald die WebResource ein Objekt zurückgibt
    - Hier wird das Objekt in die gewünschte Repräsentation konvertiert
  - **getSize()**
    - Wird vor writeTo aufgerufen um die Grösse zu ermitteln
    - Sollte die Grösse vor der Konvertierung nicht bekannt sein, kann ein -1 rückgegeben werden
  - **isWriteable()**
    - Prüft ob dieser Writer für das angeforderte Format eingesetzt werden soll

EXAMPLE - MESSAGEBODYWRITER

```
@Provider
@Produces("text/csv")
public class CSVWriter implements MessageBodyWriter<List<Customer>> {

    @Override
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations, MediaType mediaType)
{
        if (mediaType.getType().equals("……")
                && mediaType.getSubtype().matches("……")) {
            return true;
        }
        return false;
    }

    @Override
    public long getSize(List<Customer> customers, Class<?> type,
            Type genericType, Annotation[] annotations, MediaType mediaType) {
        return -1;
    }

    @Override
    public void writeTo(List<Customer> customers, Class<?> type,
            Type genericType, Annotation[] annotations, MediaType mediaType,
            MultivaluedMap<String, Object> httpHeaders, OutputStream stream) {
        PrintWriter writer = new PrintWriter(stream);
        for (Customer customer : customers) {
            writer.println(……);
        }
        writer.flush();
    }
}
```

EXAMPLE - MESSAGEBODYREADER

```
@Provider
@Consumes("text/csv")
public class CSVReader implements MessageBodyReader<List<Customer>> {

   @Override
   public boolean isReadable(Class<?> type, Type genericType,
          Annotation[] annotations, MediaType mediaType) {
      // TODO Auto-generated method stub
      return null;
   }

   @Override
   public List<Customer> readFrom(Class<List<Customer>> customers,
          Type genericType, Annotation[] annotations, MediaType mediaType,
          MultivaluedMap<String, String> httpHeaders, InputStream stream)
          throws IOException, WebApplicationException {
      // TODO Auto-generated method stub
      return null;
   }
}
```


RESTFUL WEB SERVICE CLIENT

- Jersey ist die Referenz-Implementierung der REST Spezifikation in Java
- Jersey besteht im wesentlichen aus einem REST server und einem REST client.
- Die Client Application erstellt eine Instanz welche mit der WebResource verbunden ist und kommuniziert mittels HTTP Requests mit dieser

JERSEY MAVEN

```
<dependency>
   <groupId>org.glassfish.jersey.core</groupId>
   <artifactId>jersey-client</artifactId>
   <version>${jersey.version}</version>
</dependency>

<dependency>
   <groupId>org.glassfish.jersey.core</groupId>
   <artifactId>jersey-server</artifactId>
   <version>${jersey.version}</version>
</dependency>
```


RESTFUL WEB SERVICE CLIENT

```
// Import
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;

// Client
private Client client = ClientBuilder.newClient();
private WebTarget webResource = client.target("URI");;


// GET-Methode
Customer customer = webResource.request().get(Customer.class);

// GET-Methode with path (param)
Customer customer =
webResource.path("path").request(MediaType.APPLICATION_XML).get(Customer.class);
```

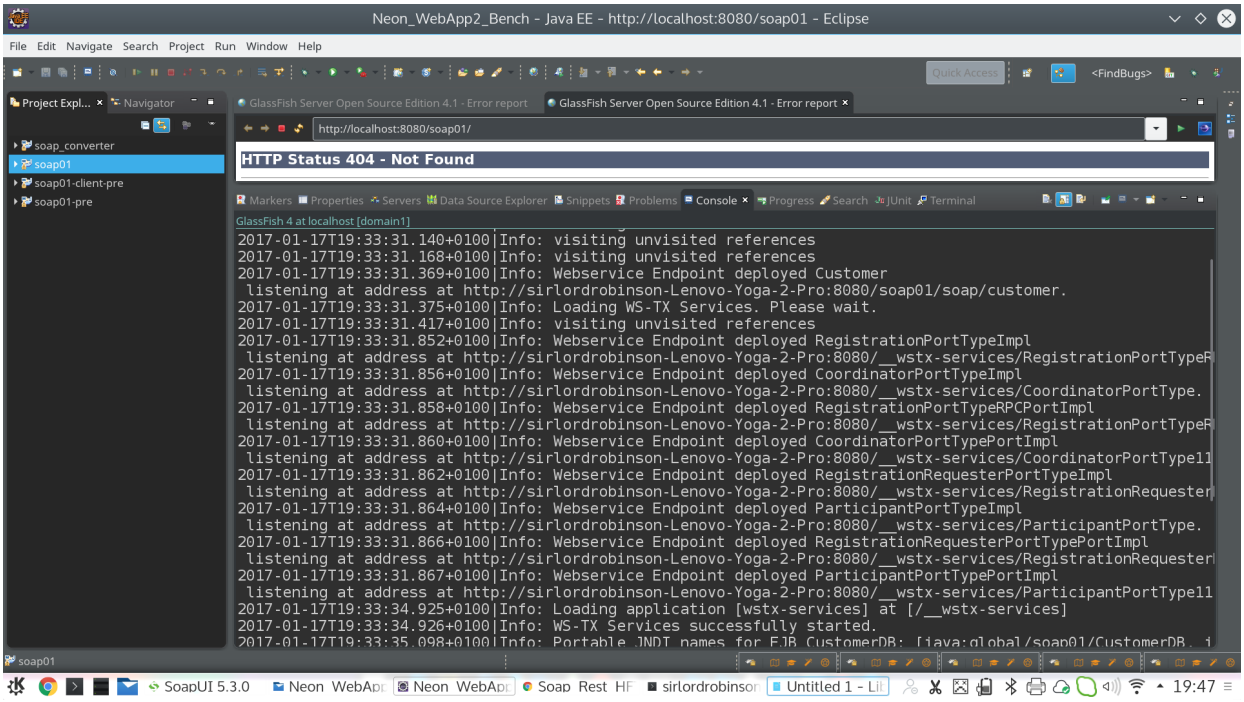
REST Exercise 02: Client / Exception / MessageBodyWriter

- Project rest02 erstellen
- Alle Rest URI's sollen unter rest02/rest/*** erreichbar sein
  - mapping auf /rest/* setzen (Deployment Descriptor)
- Rückgabe der Methode *addCustomer* von Customer auf Response umstellen
  - Die Response soll den HTTP Status Created sowie den Customer übermitteln
- Methode searchCustomers() implementieren
  - path: byName
  - QueryParam für first- und last -Name erstellen
  - Exception NotFoundException erstellen und gegebenenfalls "werfen"
  - Exception BadRequestException erstellen und mit MissingSearchCriteriaException "mappen" (catch -> throw new)
  - siehe dir dazu die *findCustomer* Methode der *customerDB*Klasse an
- MessageBodyWriter Klasse für text/csv erstellen
  - bei Methode getAllCustomers text/csv als zusätzlichen Rückgabewert definieren
  - Mittels RESTClient testen (Header: Key=Accept , Value=text/csv)
- rest02-client-pre importieren
  - CustomerBaseRestProxy Klasse implementieren
  - Mittels CustomerBaseClient testen (RunAs Java Application)
- Erweitert:
  - Methode delete / update
  - DB Anbindung
  - JSF Client entwickeln und rest Service anbinden (GUI)

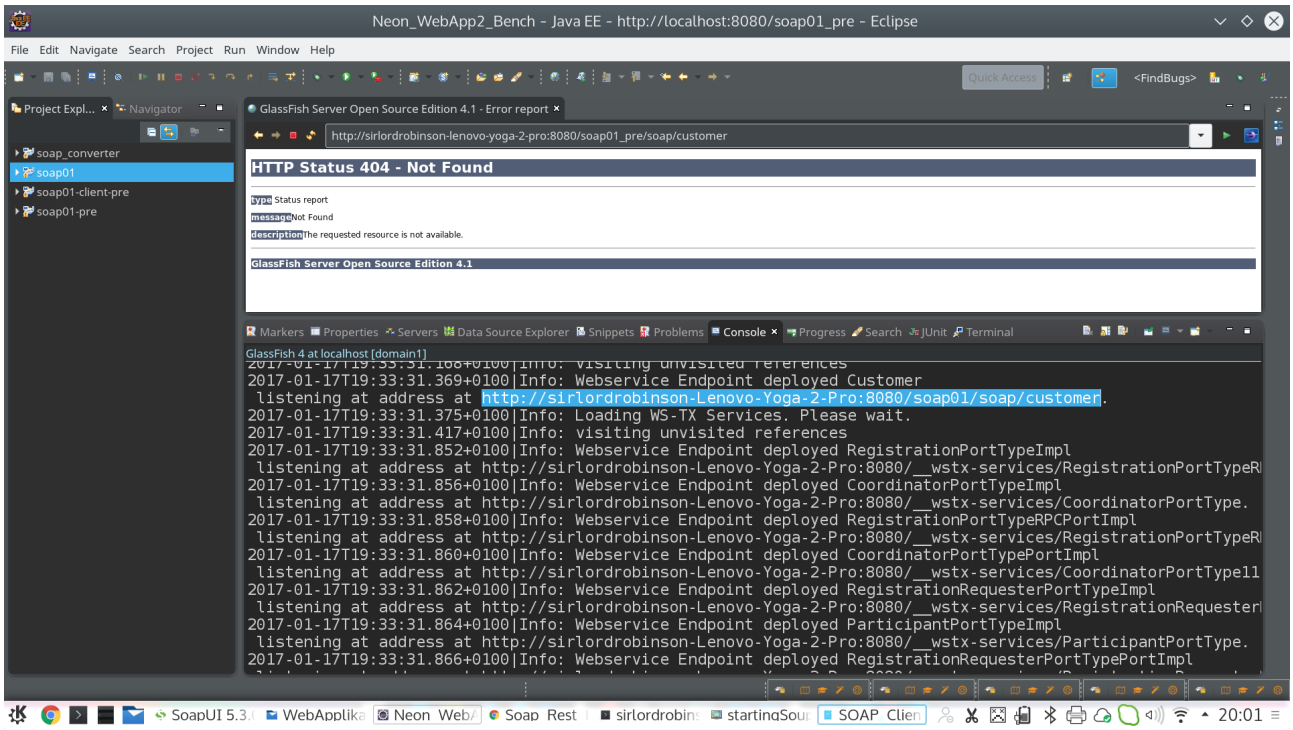
SOAP Exercise 02: Client / wsimport

Steps

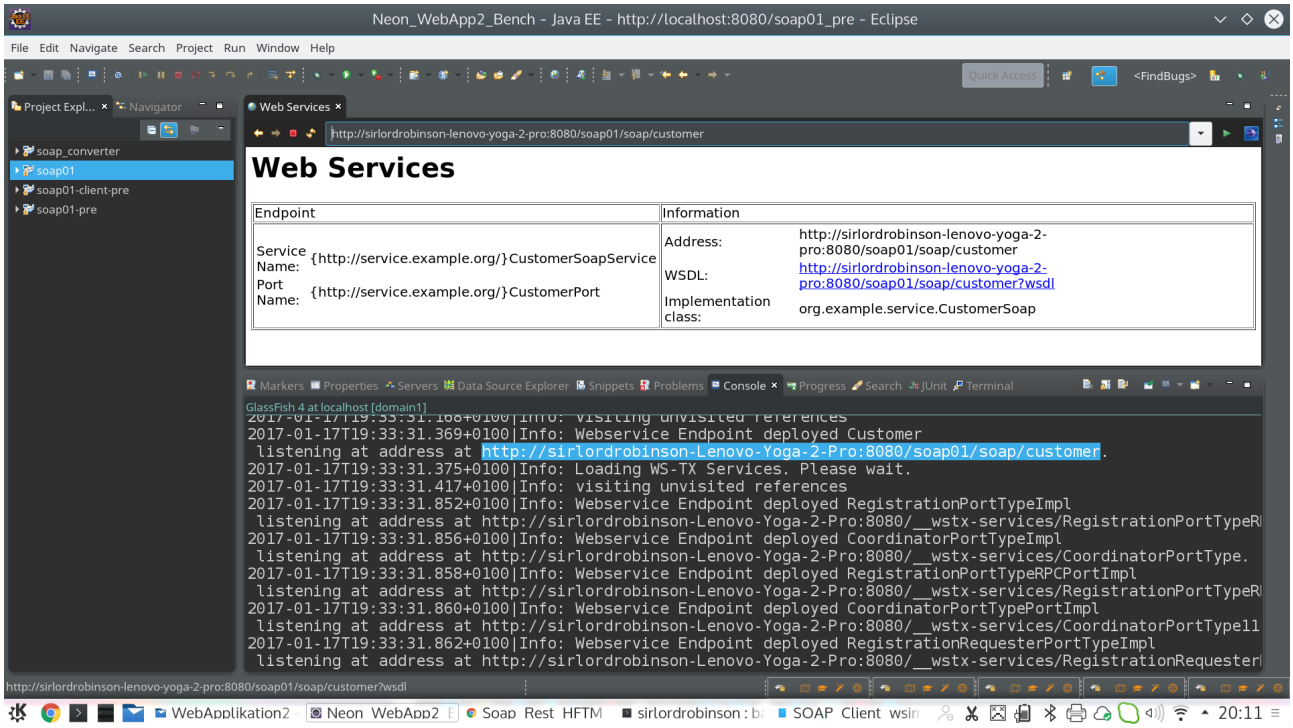
1. Right click on the projekt package that contains the classes that will be generated from, here is **Soap01**, Run As, Run on server. deployed



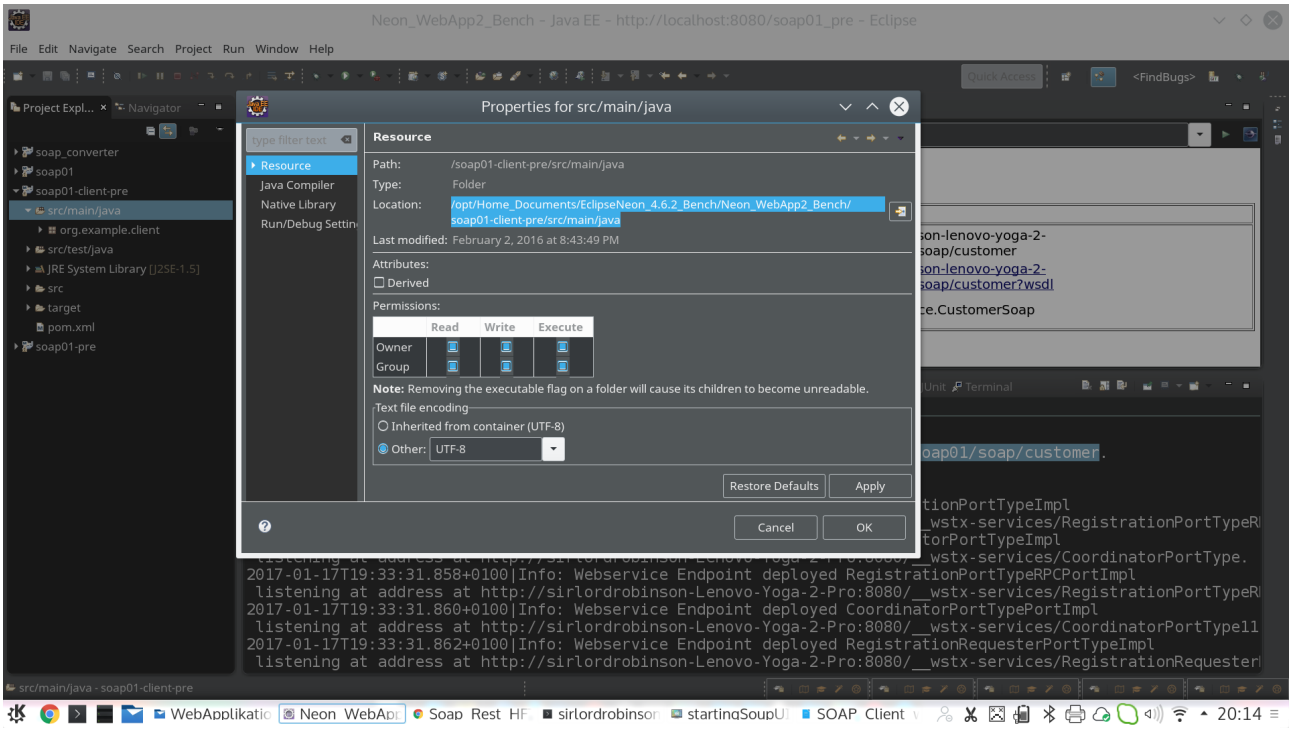
2. Copy the highlighted text and paste it on the eclipse browser.



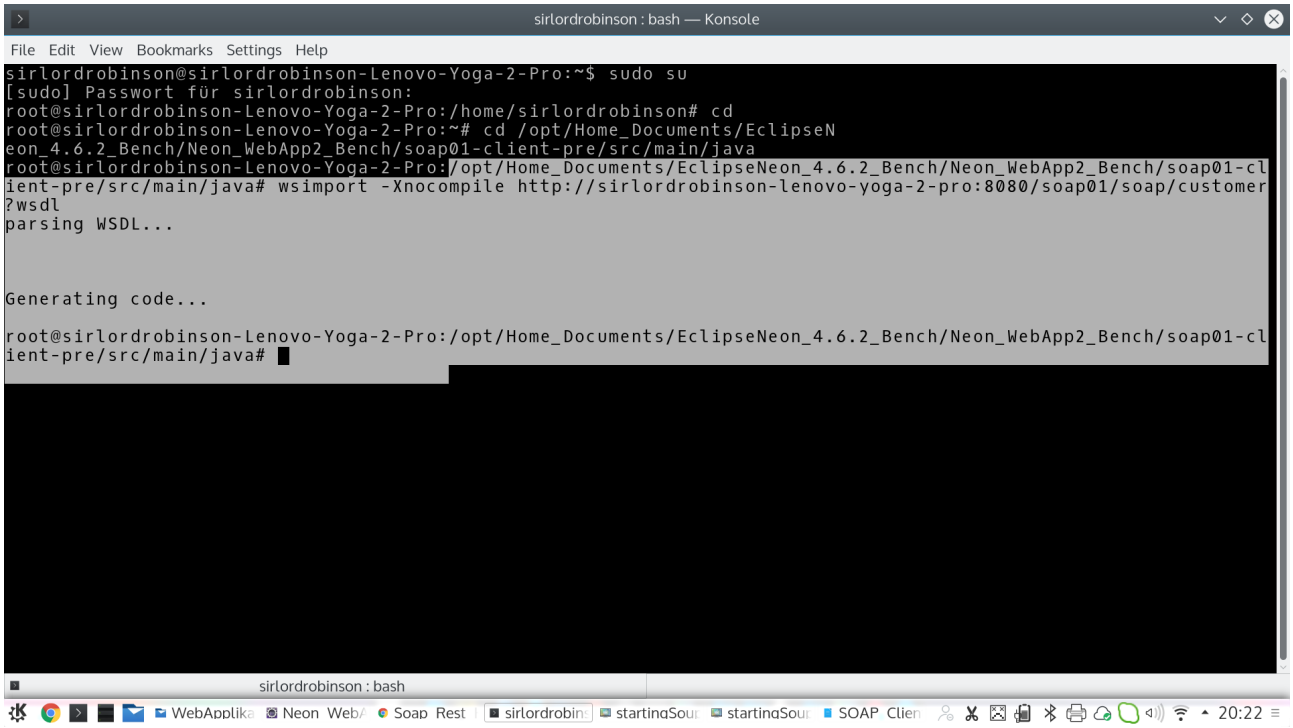
3. After pasting the text on the browser, Press the enter botton, the WSDL addresse will be generated automatically.



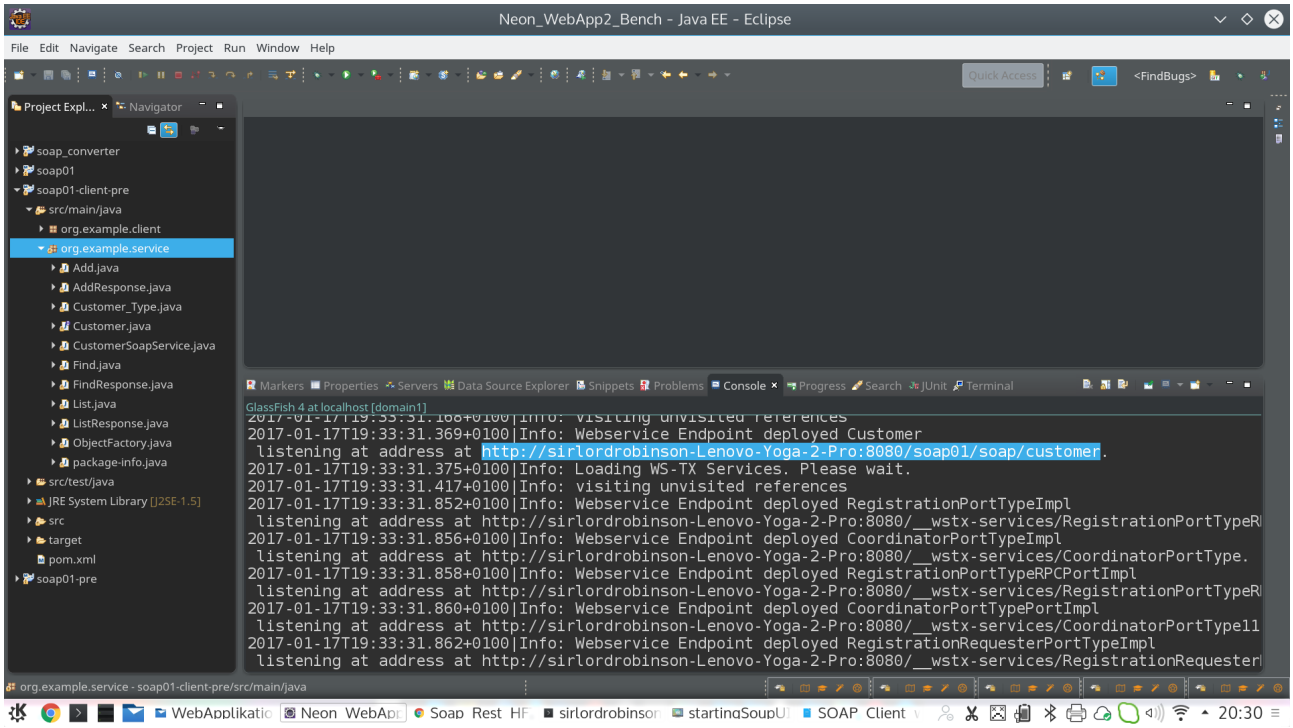
4. Choose where to store the generated classes, then from the command line navigate to that location. Eg my location is: /opt/Home_Documents/EclipseNeon_4.6.2_Bench/Neon_WebApp2_Bench/soap01-client-pre/src/main/java



Then go to command line, navigate to the location where the artifacts will be located.

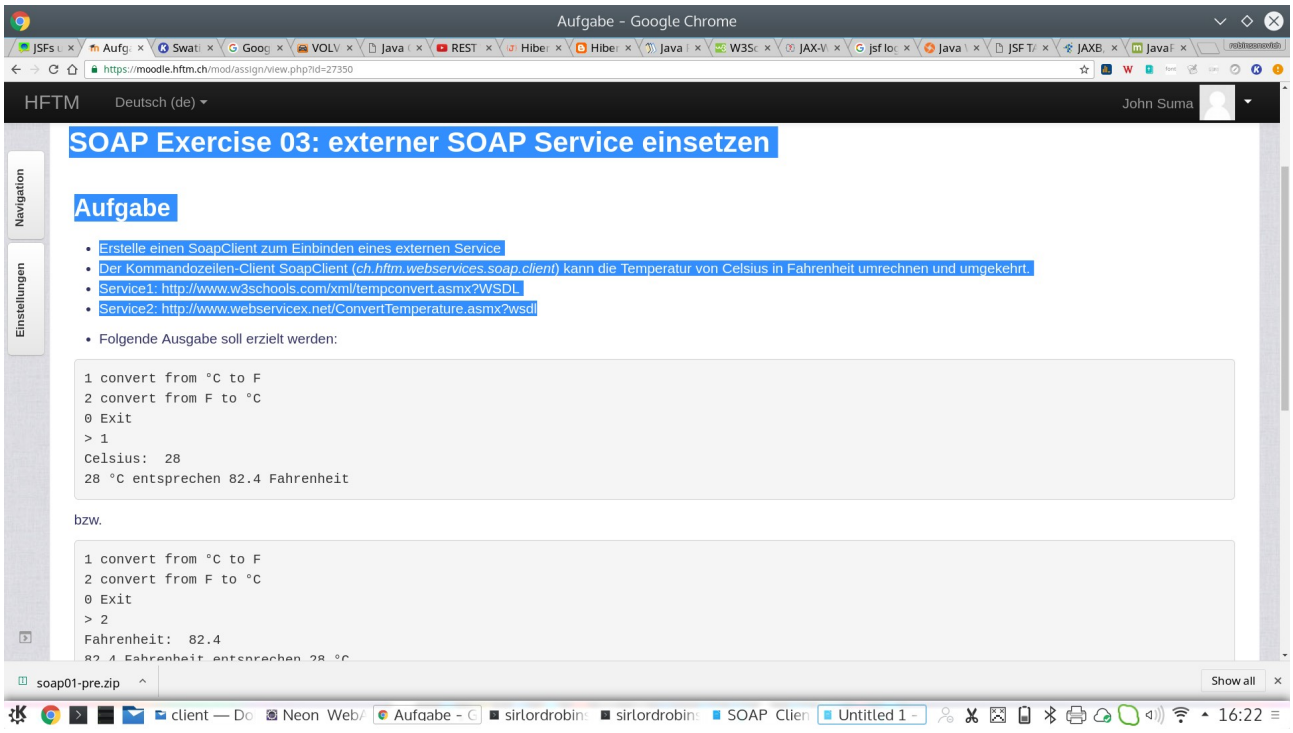


Go to your eclipse project and refresh, by doing file, refresh.
Then the generated classes should be there in your location.
Here it is located in project : **Soap01-client-pre**
Package: org.example.service



```
import java.util.Scanner;

public class CustomerBaseClient {

	private static Scanner scanner;

	public static void main(String[] args) {

		scanner = new Scanner(System.in);
		while (true) {
			System.out.println("1 List contacts");
			System.out.println("2 Find contact");
			System.out.println("3 Add contact");
			System.out.println("4 Update contact");
			System.out.println("5 Remove contact");
			System.out.println("6 Exit");
			System.out.print("> ");
			try {
				int action = Integer.parseInt(scanner.nextLine());
				if (action == 1) {
					listCustomer();
				} else if (action == 2) {
					findCustomer();
				} else if (action == 3) {
					addCustomer();
				} else if (action == 4) {
					updateContact();
				} else if (action == 5) {
					removeContact();
				} else if (action == 6) {
					System.exit(0);
				}
			} catch (NumberFormatException e) {
				System.err.println("Invalid input");
			} catch (Exception e) {
				System.err.println("Unexpected system error");
			}
			System.out.println();
		}
	}

	private static void listCustomer() throws Exception {
		System.out.print("todo: listCustomer()");
	}

	private static void findCustomer() throws Exception {
		System.out.print("todo: findCustomer()");

	}

	private static void addCustomer() throws Exception {
		System.out.print("todo: addCustomer()");
```

```
}

private static void updateContact() {
        System.out.print("todo: updateContact()");
}

private static void removeContact() {
        System.out.print("todo: removeContact()");
}
```




```
public class SoapClient {

        private static Scanner scanner;
        // Step 1 Deklaration Stub
        private static TempConvertSoap tempConvertSoap;


        public static void main(String[] args) {
                try {

                        // step2 serviceFactory erstellen
                        TempConvert serviceFactory = new TempConvert();


                        // step3 stub obj aus ServiceFactory
                        tempConvertSoap = serviceFactory.getTempConvertSoap();



                } catch (Exception e) {
                        System.out.println(e.getMessage());
                        System.out.println("Client wird beendet");
```

```
                System.exit(0);
        }

        scanner = new Scanner(System.in);
        while (true) {
                System.out.println("1 convert from °C to F");
                System.out.println("2 convert from F to °C");
                System.out.println("0 Exit");
                System.out.print("> ");
                try {
                        int action = Integer.parseInt(scanner.nextLine());
                        if (action == 1) {
                                convertCelsiusToFahrenheit();
                        } else if (action == 2) {
                                convertFahrenheitToCelsius();
                        } else if (action == 0) {
                                System.exit(0);
                        }
                } catch (NumberFormatException e) {
                        System.err.println("Invalid input");
                } catch (Exception e) {
                        System.err.println("Unexpected system error");
                }
                System.out.println();
        }
}


private static void convertCelsiusToFahrenheit() throws Exception {
        System.out.print("Celsius:  ");
        String celsius = scanner.nextLine();
        try {
                 String fahrenheit = tempConvertSoap.celsiusToFahrenheit(celsius);
        } catch (Exception e) {
                System.out.println("Konvertierung fehlgeschlagen");
        }

}



private static void convertFahrenheitToCelsius() throws Exception {
        System.out.print("Fahrenheit:  ");
        String fahrenheit = scanner.nextLine();
        try {
                String celsius = tempConvertSoap.fahrenheitToCelsius(fahrenheit);
        } catch (Exception e) {
                System.out.println("Konvertierung fehlgeschlagen");
        }
}
```

Augabe: DeploymentDescriptor anpassen (**uri**: soap/customer)

Deployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
<servlet>
<servlet-name>Customer</servlet-name>
<servlet-class>org.example.service.CustomerSoap</servlet-class>
</servlet>  <servlet-mapping>
<servlet-name>Customer</servlet-name>
 <url-pattern>/soap/customer</url-pattern>
  </servlet-mapping>
</web-app>
```

```
@Startup
@Singleton(name = "CustomerDB")
public class CustomerDB {
	private List<Customer> customers = new ArrayList<Customer>();
	private static CustomerDB instance;

	public CustomerDB() {

		Customer a = new Customer();
		a.setFirstName("Andrea");
		a.setLastName("A");
		customers.add(a);

		Customer b = new Customer();
		b.setFirstName("Berta");
		b.setLastName("B");
		customers.add(b);
	}

	public Customer addCustomer(Customer customer) {
		customers.add(customer);
		System.out.println("CustomerDB.addCustomer");
		return customer;
	}

	public List<Customer> getAllCustomers() {
		return customers;
	}

	public Customer getCustomer(int index) {
		return customers.get(index);
	}

	public List<Customer> findCustomer(String firstName, String lastName)
			throws MissingSearchCriteriaException {
		if ((firstName == null || firstName.isEmpty())
				&& (lastName == null || lastName.isEmpty())) {
			throw new MissingSearchCriteriaException();
		}

		firstName = firstName.toLowerCase();
		lastName = lastName.toLowerCase();;
		List<Customer> searchResult = new ArrayList<Customer>();

		for (Customer customer : customers) {
			if (customer.getFirstName().toLowerCase().indexOf(firstName) >= 0
					&&
customer.getLastName().toLowerCase().indexOf(lastName) >= 0) {
				searchResult.add(customer);
			}
		}
		return searchResult;
```

```
	}

	public static CustomerDB getInstance() {
		if (instance == null) {
			instance = new CustomerDB();
		}
		return instance;
	}
```

```
@Singleton(name = "WeatherStation")
public class WeatherStationBean implements WeatherStationRemote{

	@Resource
	private TimerService timerService;

@Override
	public void calculateWeatherData() {
		timerService.createIntervalTimer("Montag", "Mittwoch", "Samstag");
		ScheduleExpression calculate = new ScheduleExpression();
		calculate.second(20);
		System.out.println("calculateWeatherData");

	}

}
```

```
@WebService(name = "Customer")
public class CustomerSoap {

   @EJB
   private CustomerBean customerBean;

   @WebMethod(operationName = "list")
   @WebResult(name = "customer")
   public List<Customer> list(@WebParam(name = "max") int max) {
      return customerBean.getAllCustomers();
   }

   @WebMethod(operationName = "add")
   @WebResult(name = "customer")
   public Customer add(@WebParam(name = "customer") Customer customer){
       return customerBean.addCustomer(customer);
   }

   @WebMethod(operationName = "find")
```

```
    @WebResult(name = "customer")
    public List<Customer> find(@WebParam(name = "firstName") String firstName,
@WebParam(name = "lastName") String lastName)
         throws NotFoundException {
        List<Customer> customers = null;
        try {
            customers = customerBean.searchCustomers(firstName, lastName);
        }catch (MissingSearchCriteriaException e){
            e.printStackTrace();
        }
        return customers;
    }
```