Component-Based Software Engineering
Dipl.-Inf. Sven Karol
Office Hours: on demand
INF 2084
sven.karol@tu-dresden.de
`http://st.inf.tu-dresden.de/teaching/cbse`

Exercise Sheet No. 5
Software Technology Group
Institute for Software and Multimedia Technology
Department of Computer Science
Technische Universität Dresden
01062 Dresden

Web Services

Task 1: Developing a Simple Web Service

In this task, we create a simple SOAP-based web service using the JAX-WS library bundled with JDK/JRE $\geq$ 1.6.

1a) **Task:** Consider a car rental company which lends cars of different *quality* to their customers. Imagine, you have been hired by that company to improve their software. Assume the car rental company already has some Java-based standalone application to manage the rentals, but now they want to expand their business by setting up a new online store and open up to market places like booking.com. Therefore, the company wants you to implement some *web services* to safely expose their data and to reuse some of the components of the original stand-alone application.

How can you do this using JAX-WS?

Solution: JAX-WS provides *Annotations* to mark interfaces and implementing classes as web services, which can be queried and executed via the SOAP http protocol. Each annotated web service class is registered under service URL and provides an automatically generated WSDL interface description.

1b) **Task:** Consider the following Java class in the standalone system:

```
public class CarManagementImpl implements CarManagement {

  //Map that contains a mapping from class to car name
  private Map<String,String> carDataBase = new HashMap<String,String>();
  {
    carDataBase.put("1", "Fiat Croma");
    carDataBase.put("2", "BMW 118i");
    carDataBase.put("3", "VW Pheaton");
    carDataBase.put("10", "Maybach Landaulet");
  };

  //Returns a car name for the requested class, or null
  public String requestCar(String clazz) {
    return carDataBase.get(clazz);
  }
}
```

Use JAX-WS to specify and run a web service for that class! For testing, make the service available from `http://localhost:9009/cars`. Inspect the WSDL description under `http://localhost:9009/cars?wsdl` using a standard web browser and compare it to the actual interface of the original CarManagement class.

Solution:

```java
package webservices.tud.org;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

/**
 * Interface of the car management service.
 *
 * @author skarol
 */
@WebService(name = "CarManagement", serviceName = "CarManagementService")
@SOAPBinding(style = Style.RPC)
public interface CarManagement {

  @WebMethod public abstract String requestCar(String clazz);

}
```

—

```java
package webservices.tud.org;

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

/**
 * Car management implementation.
 *
 * @author skarol
 */
@WebService(endpointInterface = "webservices.tud.org.CarManagement")
public class CarManagementImpl implements CarManagement {

  private Map<String,String> carDataBase = new HashMap<String,String>();
  {
    carDataBase.put("1", "Fiat Croma");
    carDataBase.put("2", "BMW 118i");
    carDataBase.put("3", "VW Pheaton");
    carDataBase.put("10", "Maybach Landaulet");
  };

  public String requestCar(String clazz) {
    return carDataBase.get(clazz);
  }
}
```

—

```java
package webservices.tud.org;

import javax.xml.ws.Endpoint;


/**
 * Run this class to publish the web service.
 *
 * @author skarol
 */
public class Publisher {

  public static void main(String[] args) {
    Endpoint.publish("http://localhost:9009/cars", new CarManagementImpl());
  }

}
```

—

After compilation, you can run the `Publisher` class. The WSDL generated from `http://localhost:9009/cars?wsdl` looks as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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://org.tud.webservices/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://org.tud.webservices/"
  name="CarManagementImplService">
  <types></types>
  <message name="requestCar">
    <part name="arg0" type="xsd:string"/>
  </message>
  <message name="requestCarResponse">
    <part name="return" type="xsd:string" />
  </message>
  <portType name="CarManagement">
    <operation name="requestCar">
      <input
         wsam:Action="http://org.tud.webservices/CarManagement/requestCarRequest"
         message="tns:requestCar"/>
      <output
         wsam:Action="http://org.tud.webservices/CarManagement/requestCarResponse"
         message="tns:requestCarResponse"/>
    </operation>
  </portType>
  <binding name="CarManagementImplPortBinding" type="tns:CarManagement">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="requestCar">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal" namespace="http://org.tud.webservices/" />
      </input>
      <output>
        <soap:body use="literal" namespace="http://org.tud.webservices/" />
      </output>
    </operation>
  </binding>
  <service name="CarManagementImplService">
    <port name="CarManagementImplPort" binding="tns:CarManagementImplPortBinding">
      <soap:address location="http://localhost:9009/cars" />
    </port>
  </service>
</definitions>
```

$\boxed{\text{1c)}}$ **Task:** Call the web service via SOAP-HTTP requests. Use JAX-WS to generate Proxies and Stubs for accessing your service. Describe how they work!

Solution: You can use the `javax.xml.ws.Service` class to call a webservice. Calling the car management service looks as follows:

```java
package webservices.client.tud.org;

import java.io.IOException;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;

import webservices.tud.org.CarManagement;

/**
 * This class demonstrates a simple web service client
 *
 * @author skarol

 */
public class CarManagementSimpleClient {
  public static void main(String[] args) throws IOException {
    //URL for loading the descriptor
    java.net.URL url = new java.net.URL("http://localhost:9010/cars?wsdl");

    //create a qualified name object for the service that we want to use
    QName name = new QName("http://org.tud.webservices/", "CarManagementImplService");

    //create descriptor for remote service
    Service carManagementService = Service.create(url, name);

    //creates CarMangement proxy object
    CarManagement mgmt = carManagementService.getPort(CarManagement.class);

    //request a car of class 2
    String result = mgmt.requestCar("2");

    //print out the result
    System.out.println("You will get a " + result);
  }
}
```

The http GET request for the WSDL service descriptor:

```
GET /cars?wsdl HTTP/1.1
User-Agent: Java/1.7.0_21
```

```
Host: localhost:9009
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

The http POST message with the SOAP request envelope generated by the JAX-WS proxy:

```
POST /cars HTTP/1.1
Accept: text/xml, multipart/related
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://org.tud.webservices/CarManagement/requestCarRequest"
User-Agent: JAX-WS RI 2.2.4-b01
Host: localhost:9009
Connection: keep-alive
Content-Length: 203

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:requestCar xmlns:ns2="http://org.tud.webservices/">
      <arg0>2</arg0>
    </ns2:requestCar>
  </S:Body>
</S:Envelope>
```

The http message with the SOAP response generated by the Server:

```
HTTP/1.1 200 OK
Transfer-encoding: chunked
Content-type: text/xml; charset=utf-8
Date: Mon, 17 Jun 2013 10:48:32 GMT

5e
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:requestCarResponse xmlns:ns2="http://org.tud.webservices/">
      <return>BMW 118i</return>
    </ns2:requestCarResponse>
  </S:Body>
</S:Envelope>
0
```

Task 2: Examining Web Services 1

In this task we are going to use an existing web service to query some data from it. The *jExam* project provides web service access to all public information in *jExam*. In this exercise, you are going to inspect some of that data. As seen in the previous task, it would be a mess to query a web service with SOAP envelopes manually. Hence, we use the JAX-WS API again and the `wsimport` tool bundled with JDK 1.7. To install the tool, you have to download and install a JDK 1.7 (or higher), such that `JAVA_HOME` points to the path of the JDK.

2a) **Task:** First, inspect the WSDL of *jExam* using a web browser. You can find it at `http://jexam.inf.tu-dresden.de:9009/WebServiceJBoss/JExamWebServiceBean?wsdl`.

Solution: Here is the part of the WSDL which is relevant for the next subtask (we skip the rest because of the length of the service description):

```
<definitions name="JExamWebService" targetNamespace="http://webservice.jexam.de/">
  <types>
    <xs:schema targetNamespace="http://webservice.jexam.de/" version="1.0">
      <xs:element name="Exception" type="tns:Exception"/>
      <xs:element name="getCurrentSemester" type="tns:getCurrentSemester"/>
      <xs:element name="getCurrentSemesterResponse" type="tns:getCurrentSemesterResponse"/>
      <xs:element name="getOrganisationUnitsByAbbreviation" type="
          tns:getOrganisationUnitsByAbbreviation"/>
      <xs:element name="getOrganisationUnitsByAbbreviationResponse" type="
          tns:getOrganisationUnitsByAbbreviationResponse"/>
      <xs:element name="getTeachingOffersByOrganisationUnitAndSemester" type="
          tns:getTeachingOffersByOrganisationUnitAndSemester"/>
      <xs:element name="getTeachingOffersByOrganisationUnitAndSemesterResponse" type="
          tns:getTeachingOffersByOrganisationUnitAndSemesterResponse"/>
      <xs:complexType ....
    </xs:schema>
  </types>
  <message name="JExamWebServiceBean_getCurrentSemester">
    <part element="tns:getCurrentSemester" name="getCurrentSemester"/>
  </message>
  <message name="JExamWebServiceBean_getTeachingOffersByOrganisationUnitAndSemester">
    <part element="tns:getTeachingOffersByOrganisationUnitAndSemester" name="
        getTeachingOffersByOrganisationUnitAndSemester"/>
  </message>
  <message name="JExamWebServiceBean_getOrganisationUnitsByAbbreviation">
    <part element="tns:getOrganisationUnitsByAbbreviation" name="
        getOrganisationUnitsByAbbreviation"/>
  </message>
  <message name="Exception">
    <part element="tns:Exception" name="Exception"/>
  </message>
  <portType name="JExamWebServiceBean">
    <operation name="getCurrentSemester" parameterOrder="getCurrentSemester">
      <input message="tns:JExamWebServiceBean_getCurrentSemester"/>
      <output message="tns:JExamWebServiceBean_getCurrentSemesterResponse"/>
      <fault message="tns:Exception" name="Exception"/>
      <jaxws:bindings>
        <jaxws:method name="getCurrentSemester"></jaxws:method>
      </jaxws:bindings>
    </operation>
    <operation name="getOrganisationUnitsByAbbreviation" parameterOrder="
        getOrganisationUnitsByAbbreviation">
              <input message="tns:JExamWebServiceBean_getOrganisationUnitsByAbbreviation"/>
              <output message="
                  tns:JExamWebServiceBean_getOrganisationUnitsByAbbreviationResponse"/>
              <fault message="tns:Exception" name="Exception"/>
    </operation>
    <operation name="getTeachingOffersByOrganisationUnitAndSemester" parameterOrder="
        getTeachingOffersByOrganisationUnitAndSemester">
      <input message="tns:JExamWebServiceBean_getTeachingOffersByOrganisationUnitAndSemester
          "/>
      <output message="
          tns:JExamWebServiceBean_getTeachingOffersByOrganisationUnitAndSemesterResponse"/>
      <fault message="tns:Exception" name="Exception"/>
    </operation>
  </portType>
  <binding name="JExamWebServiceBeanBinding" type="tns:JExamWebServiceBean">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getCurrentSemester">....</operation>
    <operation name="getOrganisationUnitsByAbbreviation">....</operation>
    <operation name="getTeachingOfferCommentariesByOrganisationUnitAndSemester">....</
        operation>
  </binding>
  <service name="JExamWebService">
    <port binding="tns:JExamWebServiceBeanBinding" name="JExamWebServiceBeanPort">
```

```
      <soap:address location="http://jexam.inf.tu-dresden.de:9009/WebServiceJBoss/
          JExamWebServiceBean"/>
    </port>
  </service>
</definitions>
```

2b) **Task:** Use the `wsimport` tool from console to create a stub for the latest version of the jExam web service via (NOTE: option `-d` sets the target directory, `-Xnocompile` disables compilation and emits Java sources):

```
"%JAVA_HOME%/bin/wsimport" -d src-gen -Xnocompile
   http://jexam.inf.tu-dresden.de:9009/WebServiceJBoss/JExamWebServiceBean?wsdl
```

Inspect the generated Java files. How is the Java API for XML bindings (JAXB) used? Now use the generated stub to obtain information on all courses offered by the ST group (abbreviated 'SWT' in jExam) in the current semester. Print out a list with information on each course and the staff member responsible for it.

Solution: JAXB is used as *native* XML binding for the embedded schema types in the WSDL specification.

Here is my programmed solution:

```
package se.jexam.webservice.client;

import java.util.List;

import de.jexam.webservice.*;

/**
 * A simple program to query the jExam web service using the bindings
 * generated by JAX-WS.
 *
 * @author skarol
 */
public class InspectorJexam {

  public static void main(String[] args) throws Exception_Exception {
    JExamWebService service = new JExamWebService();
    JExamWebServiceBean bean = service.getJExamWebServiceBeanPort();

    WsSemester semester = bean.getCurrentSemester();
    log("Current semester: " + semester.getName() + ", ID:"+semester.getId());

    List<WsOrganisationUnit> units =
      bean.getOrganisationUnitsByAbbreviation("SWT",true);
    WsOrganisationUnit unit = units.get(0);
    log("Unit " + unit.getAbbreviation() + " is " +
            unit.getName() + ", ID:"+unit.getId());

    List<WsTeachingOffer> offers =
        bean.getTeachingOffersByOrganisationUnitAndSemester
             (unit.getId(),semester.getId());
    log("Unit " + unit.getAbbreviation() + " has the following offers:");
    for(WsTeachingOffer offer:offers) {
      WsSubject subject = offer.getSubject();
      log("\tName:" + subject.getName());
      log("\tShort:" + subject.getAbbreviation());
      log("\tID:" + subject.getId());
      log("\tECTS_ID:" + subject.getEctsNo());
      log("\tSWS:(" + subject.getSppwOfLecture() + ","
          + subject.getSppwOfPracticalCourse() + ")");
      String url = "".equals(subject.getUrl())?"No Website available":subject.getUrl();
      log("\tVisit website at: " + url);
      log("");
    }
  }

  public static void log(String s){
    System.out.println(s);
  }
}
```

Task 3: Examining Web Services 2

The discogs project at http://discogs.com provides web service access to all public information in discogs. In this exercise, you are going to inspect some of that data.

3a) **Task:**

Read and understand the Discogs API at http://www.discogs.com/developers/.

How does this web service differ from the jExam one? Which methods are provided via api.discogs.com, how are those methods called and in which format are results passed to the client?

Solution: Discogs is a *RESTful* web service. REST stands for *Representational State Transfer* and strictly relies on the methods of HTTP and supported MIME types. REST is a *stateless* and has a uniform API.Discogs uses `application/json` as content type. JSON stands for *Javascript Object Notation*.

3b) **Task:**

Use the web service to list all releases of the label *Pampa Germany* and provide a detailed view (including track data) of *Robag Wruhme's* latest album *Thora Vukk*.

Solution: Here is my solution in Java using plain HTTP requests:

```
package discogs.client.tud;

import java.io.*;
import java.net.*;
import java.util.*;
import java.util.regex.*;
```

```
/**
 * A very simple accessory class for the Discogs RESTful service.
 *
 * @author skarol
 *
 */
public class DiscogsAccess {

  public static final Map<String,String> httpParams = new HashMap<String,String>();
  {
    httpParams.put("User-Agent", "CBSEExercise/0.1␣+http://st.inf.tu-dresden.de/");
    httpParams.put("Content-Type", "application/json");
  }

  private static final Pattern releasesPattern =
      Pattern.compile("\"releases_url\"\\s*:\\s*\"(\\S+)\"");
  private static final Pattern masterPattern =
      Pattern.compile("\"Thora␣Vukk\".+?\"resource_url\"\\s*:\\s*\"(\\S+)\"");

  public static void main(String[] args) throws IOException {
    //GET JSON representation of label
    String label = doGET("http://api.discogs.com/label/Pampa␣Germany");
    //extract request URL
    Matcher matcher = releasesPattern.matcher(label);
    String releasesURL = matcher.find()?matcher.group(1):"ERROR";
    //GET and print the releases of Pampa Germany
    String releases = doGET(releasesURL);
    log("Releases␣of␣Pampa␣Germany:");
    log(releases);

    //GET JSON representation of artist
    String artist = doGET("http://api.discogs.com/artist/Robag%20Wruhme");
    matcher = releasesPattern.matcher(artist);
    releasesURL =  matcher.find()?matcher.group(1):"ERROR";
    releases = doGET(releasesURL);
    log("Releases␣of␣Robag␣Wruhme:");
    log(releases);

    //GET JSON representation of album
    matcher = masterPattern.matcher(releases);
    String mastersURL = matcher.find()?matcher.group(1):"ERROR";
    String master = doGET(mastersURL);
    log("Information␣on␣Thora␣Vukk␣:");
    log(master);
  }

  /**
   * Perform HTTP GET.
   */
  public static String doGET(String query) throws IOException{
    URL url = new URL(query );

    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    connection.setRequestMethod("GET");
    for(String key:httpParams.keySet()){
      connection.addRequestProperty(key,httpParams.get(key));
    }

    connection.connect();
    BufferedReader reader = new BufferedReader
        (new InputStreamReader((InputStream) connection.getContent()) );
    String result = ””;
    for ( String line; (line = reader.readLine()) != null; ) {
      result += line +”\n”;
    }
    connection.disconnect();

    return result;
  }

  public static void log(String s){
    System.out.println(s);
  }
}
```

—

Result of the first query:

```
{"pagination":
    {"per_page": 50,
  "items": 1,
  "page": 1,
  "urls": {},
  "pages": 1},
 "releases":
      [{"status": "Accepted",
        "format": "Shellac, 10\"",
        "title": "Las Tres De La Ma\u00f1ana / Loca",
        "catno": "PS 6012",
        "resource_url": "http://api.discogs.com/releases/3192576",
        "artist": "Enrique Mora Y Su Cuarteto Tipico",
        "id": 3192576}]}
```

NOTE: In a real world scenario, it is recommended to use a JSON binding like `http://json.org/java/` instead of regular expressions, or to use APIs and stubs like in the SOAP examples.