

# SOAP Web-Services with JAX-WS

Introduction to Service Design and Engineering 2013/2014.

*Lab session #10*

**University of Trento**

# Outline

- JAX-WS Overview
- JAX-WS Example
- Assignment #3

# Pre-Requisites

- Download [JAX-WS RI](#) (just in case, libraries should be already available as part of java distribution)

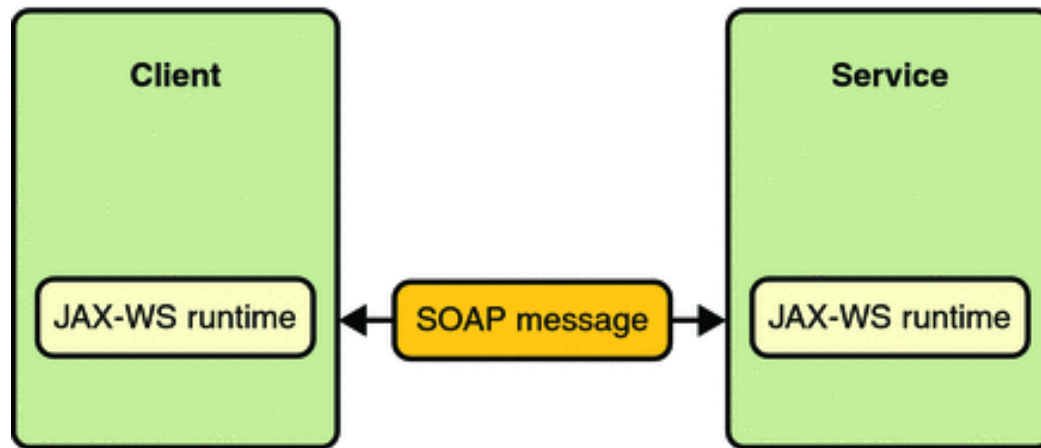
# JAX-WS Overview (1)

- JAX-WS stands for Java API for XML Web Services.
- Technology for building web services and clients that communicate using XML.
- JAX-WS allows developers to write message-oriented as well as RPC-oriented web services.
- Web service invocation is represented by an XML-based protocol such as SOAP.
- SOAP defines the envelope structure, encoding rules, and conventions for representing web service invocations and responses. \* Calls and responses are transmitted as SOAP messages (XML files) over HTTP.

## JAX-WS Overview (2)

- JAX-WS API hides SOAP's complexity from the application developer.
- On the server side, the developer specifies the web service operations by defining methods in an interface
- The developer also codes one or more classes that implement those methods.
- A client creates a proxy (a local object representing the service) and then simply invokes methods on the proxy.
- The developer does not generate or parse SOAP messages. (JAX-WS runtime system converts API calls and responses to and from SOAP messages)

## JAX-WS Overview (3)



## JAX-WS Overview (4)

- There are two ways to structure a SOAP message
- **RPC Style** web service uses the names of the method and its parameters to generate XML structures that represent a method's call stack. In the early versions of SOAP (before it was publicly published), When using RPC style, the contents of the SOAP Body must conform to a structure that indicates the method name and contains a set of parameters.
- **Document style** indicates that the SOAP body contains a XML document which can be validated against pre-defined XML schema document. When using Document style, you can structure the contents of the SOAP Body any way you like.
- The response message has a similar structure containing the return value and any output parameters.

## JAX-WS Overview (5)

- For example, you can pass a **purchase order** as a document or as a parameter in a method called placeOrder.
- Document style:

```
<env:Body>  
  <m:purchaseOrder xmlns:m="someURI"> [purchase order document] </m:purchaseOrder>  
</env:Body>
```

# JAX-WS Tutorial - RPC Style (1)

- Create a Java Project (in eclipse, let's make it a Web Dynamic Project)
- Create a package named **introsde.ws**
- Create a *Web Service Endpoint Interface*
  - A web service endpoint is a service that is published for users to access
  - The web service client is the party who access the published service
  - In this case, a **HelloWorld** java interface like the following

```
package introsde.ws;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{
    @WebMethod String getHelloWorldAsString(String name);
}
```



# JAX-WS Tutorial - RPC Style (2)

- Create the *Web Service Endpoint Implementation*

```
package introsde.ws;
import javax.ws.WebService;
//Service Implementation
@WebService(endpointInterface = "introsde.ws.HelloWorld")
public class HelloWorldImpl implements HelloWorld {
    @Override
    public String getHelloWorldAsString(String name) {
        return "Hello World JAX-WS " + name;
    }
}
```

# JAX-WS Tutorial - RPC Style (3)

- Create the package **introsde.endpoint**
- Create the *Web Service Endpoint Publisher* in this package
- Run your first JAX-WS Service as a Java application
- Test that is working by accessing: <http://localhost:6900/ws/hello?wsdl>

```
package introsde.endpoint;
import javax.xml.ws.Endpoint;
import introsde.ws.HelloWorldImpl;
//Endpoint publisher
public class HelloWorldPublisher{
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:6900/ws/hello", new HelloWorldImpl());
    }
}
```

# JAX-WS Tutorial - RPC Style (4)

- Call the service via an HTTP POST request on localhost:6900/ws/hello with body

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://ws.introsde/">
    <m:getHelloWorldAsString>
      <arg0>Pinco</arg0>
    </m:getHelloWorldAsString>
  </soap:Body>
</soap:Envelope>
```

- This should be the response

```
<?xml version="1.0" ?>
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getHelloWorldAsStringResponse
      xmlns:ns2="http://ws.introsde/">
      <return>Hello World JAX-WS Pinco</return>
    </ns2:getHelloWorldAsStringResponse>
  </S:Body>
</S:Envelope>
```

# JAX-WS Tutorial - Implementing Clients

- Create a package **introsde.client** and add the following class

```
package introsde.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import introsde.ws.HelloWorld;
public class HelloWorldClient {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:6900/ws/hello?wsdl");
        // 1st argument service URI, refer to wsdl document above
        // 2nd argument is service name, refer to wsdl document above
        QName qname = new QName("http://ws.introsde/", "HelloWorldImplService");
        Service service = Service.create(url, qname);
        HelloWorld hello = service.getPort(HelloWorld.class);
        System.out.println(hello.getHelloWorldAsString("Pinco"));
    }
}
```

# JAX-WS Tutorial - Implementing Clients - Automatic (1)

- You can also use **wsimport** to parse the wsdl file and generate client files (stub) to access the published web service.
- This is useful if you don't have the webservice interface (HelloWorld) locally available as part of some library.
- With wsimport, you will generate a local stub of the remote service that will serve you as a proxy.
- wsimport should be in JDK\_PATH/bin folder.
- Create a **my-solutions** folder on your local copy of lab10.
- From the command line, execute the following inside that new folder

```
wsimport -keep http://localhost:6900/ws/hello?wsdl
```

# JAX-WS Tutorial - Implementing Clients - Automatic (2)

- You should now have an interface and a service implementation as follows:
- File **introsde/ws/HelloWorld.java**

```
package introsde.ws;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.ws.Action;
@WebService(name = "HelloWorld", targetNamespace = "http://ws.introsde/")
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface HelloWorld {
    @WebMethod
    @WebResult(partName = "return")
    @Action(input = "http://ws.introsde/HelloWorld/getHelloWorldAsStringRequest",
        output = "http://ws.introsde/HelloWorld/getHelloWorldAsStringResponse")
    public String getHelloWorldAsString(
        @WebParam(name = "arg0", partName = "arg0")
        String arg0);
}
```

# JAX-WS Tutorial - Implementing Clients - Automatic (3)

- File `introsde/ws/HelloWorldImplService.java`

```
package introsde.ws;
import java.net.MalformedURLException;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.xml.ws.WebEndpoint;
import javax.xml.ws.WebServiceClient;
import javax.xml.ws.WebServiceException;
import javax.xml.ws.WebServiceFeature;
@WebServiceClient(name = "HelloWorldImplService",
    targetNamespace = "http://ws.introsde/",
    wsdlLocation = "http://localhost:6900/ws/hello?wsdl")
public class HelloWorldImplService extends Service
{
    private final static URL HELLOWORLDDIMPLSERVICE_WSDL_LOCATION;
    private final static WebServiceException HELLOWORLDDIMPLSERVICE_EXCEPTION;
    private final static QName HELLOWORLDDIMPLSERVICE_QNAME =
        new QName("http://ws.introsde/", "HelloWorldImplService");
    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL("http://localhost:6900/ws/hello?wsdl");
        } catch (MalformedURLException ex) {
            e = new WebServiceException(ex);
        }
        HELLOWORLDDIMPLSERVICE_WSDL_LOCATION = url;
        HELLOWORLDDIMPLSERVICE_EXCEPTION = e;
    }
}
```

# JAX-WS Tutorial - Implementing Clients - Automatic (3)

```
public HelloWorldImplService() {
    super(__getWsdLocation(), HELLOWORLDDIMPLSERVICE_QNAME);
}
public HelloWorldImplService(WebServiceFeature... features) {
    super(__getWsdLocation(), HELLOWORLDDIMPLSERVICE_QNAME, features);
}
public HelloWorldImplService(URL wsdlLocation) {
    super(wsdlLocation, HELLOWORLDDIMPLSERVICE_QNAME);
}
public HelloWorldImplService(URL wsdlLocation, WebServiceFeature... features) {
    super(wsdlLocation, HELLOWORLDDIMPLSERVICE_QNAME, features);
}
public HelloWorldImplService(URL wsdlLocation, QName serviceName) {
    super(wsdlLocation, serviceName);
}
public HelloWorldImplService(URL wsdlLocation, QName serviceName,
    WebServiceFeature... features) {
    super(wsdlLocation, serviceName, features);
}
@WebEndpoint(name = "HelloWorldImplPort")
public HelloWorld getHelloWorldImplPort() {
    return super.getPort(new QName("http://ws.introsde/", "HelloWorldImplPort"),
        HelloWorld.class);
}
@WebEndpoint(name = "HelloWorldImplPort")
public HelloWorld getHelloWorldImplPort(WebServiceFeature... features) {
    return super.getPort(new QName("http://ws.introsde/", "HelloWorldImplPort"),
        HelloWorld.class, features);
}
private static URL __getWsdLocation() {
    if (HELLOWORLDDIMPLSERVICE_EXCEPTION != null) {
        throw HELLOWORLDDIMPLSERVICE_EXCEPTION;
    }
}
```



# JAX-WS Tutorial - Implementing Clients - Automatic (4)

- To use this stub, create the following program in the file **introsde/client/HelloWorldClient.java**:

```
package introsde.client;
import introsde.ws.HelloWorld;
import introsde.ws.HelloWorldImplService;
public class HelloWorldClient{
    public static void main(String[] args) {
        HelloWorldImplService helloService = new HelloWorldImplService();
        HelloWorld hello = helloService.getHelloWorldImplPort();
        System.out.println(hello.getHelloWorldAsString("Pinco"));
    }
}
```

```
javac introsde/client/HelloWorldClient.java
java introsde/client/HelloWorldClient
```

# JAX-WS Tutorial - Document style (1)

- Create a new Web Dynamic Project
- Create the packages introsde.ws, introsde.document.client, introsde.document.endpoint, introsde.document.ws.jaxws
- Create the Web Service Endpoint Interface **HelloWorld** as follows (the only change is the SOAPBinding annotation):

```
package introsde.document.ws;
import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;
//Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL) //optional
public interface HelloWorld{
    @WebMethod String getHelloWorldAsString(String name);
}

---

## JAX-WS Tutorial - Document Style (2)

* Create the Web Service Endpoint Implementation **HelloWorldImpl.java** (no changes here)

```java
package introsde.document.ws;
import javax.jws.WebService;
//Service Implementation
@WebService(endpointInterface = "introsde.document.ws.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
    @Override
```

# JAX-WS Tutorial - Document Style - Generating Artifacts (1)

- You can use **wsgen** to generate all necessary Java artifacts (mapping classes, wsdl or xsd schema).
- Run the following command on build/classes (where the compiled classes are):

```
wsgen -keep -cp . introsde.document.ws.HelloWorldImpl
```

- It will generate two classes in build/classes/introsde/ws/jaxws,
- Copy them to your **src/introsde/ws/jaxws** folder.
- These can be seen as the equivalents to the **model** in Jersey.

# JAX-WS Tutorial - Document Style - Generating Artifacts (2)

- GetHelloWorldAsString.java

```
package introsde.document.ws.jaxws;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
@XmlRootElement(name = "getHelloWorldAsString", namespace = "http://ws.document.introsde/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getHelloWorldAsString", namespace = "http://ws.document.introsde/")
public class GetHelloWorldAsString {
    @XmlElement(name = "arg0", namespace = "")
    private String arg0;
    public String getArg0() {
        return this.arg0;
    }
    public void setArg0(String arg0) {
        this.arg0 = arg0;
    }
}
```

# JAX-WS Tutorial - Document Style - Generating Artifacts (3)

- GetHelloWorldAsStringResponse.java

```
package introsde.document.ws.jaxws;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
@XmlRootElement(name = "getHelloWorldAsStringResponse", namespace = "http://ws.document.introsde/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "getHelloWorldAsStringResponse", namespace = "http://ws.document.introsde/")
public class GetHelloWorldAsStringResponse {
    @XmlElement(name = "return", namespace = "")
    private String _return;
    public String getReturn() {
        return this._return;
    }
    public void setReturn(String _return) {
        this._return = _return;
    }
}
```

# JAX-WS Tutorial - Document Style client (3)

- Create the Web Service Client

```
package introsde.document.client;
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import introsde.document.ws.HelloWorld;
public class HelloWorldClient{
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:6901/ws/hello?wsdl");
        //1st argument service URI, refer to wsdl document above
        //2nd argument is service name, refer to wsdl document above
        QName qname = new QName("http://ws.document.introsde/", "HelloWorldImplService");
        Service service = Service.create(url, qname);
        HelloWorld hello = service.getPort(HelloWorld.class);
        System.out.println(hello.getHelloWorldAsString("Pinco"));
    }
}
```

# Assignment #3: Part 1 (1)

- Using JAX-WS, implement CRUD services for the following model including the following operations
  - readPerson(int personId) (returns the Person)
  - createPerson(Person p) (returns the personId of the new Person, or a negative number representing an error)
  - updatePerson(Person p) (returns the personId of the updated Person, or a negative number representing an error)
    - [OPTION A] If p includes changes to the "healthProfile" you can should 1) replace the current profile with a new one if no hpId is specified (in which case, the old health profile should be part of the history now) or 2) update the corresponding health profile if an hpId is given
  - deletePerson(int id) (returns 0, or a negative number representing an error)
  - updatePersonHealthProfile(int personId, HealthProfile hp) (returns the id of the health profile updated, or a negative number representing an error). If no health profile exists yet, create it. In any other case, you should only update the health profile which is given by the hpId inside hp.
  - [OPTION B] addPersonHealthProfile(int personId, HealthProfile hp) (adds a new health profile to replace the current one, which should pass to the history). When creating a new health profile, you should

## Assignment #3: Part 1 (2)

- **Extra points:** Include also the service `getHealthProfileHistory(int personId)`
- **Extra points:** Connect to a database.

// History of the health profile

```
<healthProfile-history>
  <healthProfile>
    <hpId>999</hpId>
    <date>2013-12-05</date>
    <weight>78.9</weight>
    <height>172</height>
    <steps>5000</steps>
    <calories>2120</calories>
  </healthProfile>
  <healthProfile>
    <hpId>998</hpId>
    <date>2013-11-29</date>
    <weight>null</weight>
    <height>null</height>
    <steps>6430</steps>
    <height>null</height>
  </healthProfile>
  <healthProfile>
    <hpId>1000</hpId>
    <date>2013-11-05</date>
    <weight>null</weight>
    <height>null</height>
    <steps>12083</steps>
    <height>null</height>
  </healthProfile>
</healthProfile-history>
```



## Assignment #3: Part 2

- Create a simple client that call each of this services and prints the result (using your service implementation).

# Assignment Rules

- Before submission make a zip file that includes only
  - All Java source files
  - please, do not include .class or IDE generated project files
- Rename the Zip file to: your full name + assignment\_no. for example: cristhian\_parra\_3.zip
- Submission link: [www.dropitto.me/introsde2013](http://www.dropitto.me/introsde2013)
- Password will be given and class and sent to the group
- **Deadline:** 20/december
  - On this date, we will test the services matching clients and servers

# VIVA Testing Procedures for Assignments #2 and #3 (might change)

1. Each student will be provided with a **Evaluation Document** where you find what to test and where you will be able to mark if it worked or not (the evaluation document will be a google spreadsheet given on that date)
2. Using the user/password/serverUrl we will provide on the day of the VIVA, each student will connect to the testing server via SSH (i.e., `ssh user@serverUrl`)
  - This information will be provided on next lab, so that you can already run some tests before the day of the VIVA
3. **Assignment 2:**
  - Agree with your partner who will be the client/server first.
  - In the home of the user, the server should search for his/her REST services implementation and run it
  - The implementation will be at *\$HOME/assignments/servername\_serverlastname\_2/*
  - The client must now run his/her client against the server (should be on *\$HOME/assignments/clientname\_clientlastname\_2/*)
  - The server will have 10 minutes for making quick changes on the server (or requesting quick changes on the client) to make things work.
  - The client must copy all the output of the client to a file named *servername\_serverlastname\_2\_output.txt* in the

# Assignment Evaluation

- The assignment will be evaluated in terms of:
  - Requirements satisfaction
  - Execution & Deployment
  - Code design/independence/competence
  - Submitted in time ?
  - Report (or documentation)
  - Code originality (if you choose to do it in pairs)
- Extra points are used as "recovery" you didn't finish the requirements or didn't submit in time

# Other Resources

- This lab session is heavily based on examples from [JAX-WS Tutorials online](#)
- [Oracle Java EE tutorials on JAX-WS](#)
- [SOAP Binding: difference between Document and RPC Style](#)
- Tutorial that mixes [JAX-WS with JPA](#)

