Java Web Services

Exercise book

Version 1.2

Ex1-HelloWorld

HelloWorld Web service and clients demonstartion

Exercise description:

This first exercise demonstrates the creation of a simple web service including a single method:

```
String sayHello(String name)
```

and running different clients to test it.

This exercise will be completed with the instructor.

Exercise steps:

1. Browse through the web service code (HelloWorld.java) and understand it.
2. Deploy the web service using the Ant build script.
3. Browse through the static stub client code (HelloClient.java) & DII client (HelloClient2.java).
4. Run the clients and watch the result.
5. Use a web browser to watch the WSDL created for the web service.

Additional steps:

1. Us the VBScript client provided (hello.vbs) to test the web service.

Ex2-Clients

Web service Java clients

Exercise description:

In this exercise you will write 3 web service JAX-RPC clients:

1. A static stub client
2. A dynamic proxy client
3. A DII client

The clients will be used to access a Weather web service which returns the current weather by getting the day of week as a parameter.

Exercise steps:

1. Browse through the web service code (WeatherService.java) and understand it.
2. Deploy the web service using the Ant build script.
3. Write a static stub client, remember that before writing the code, a stub must be created using the Ant script.
4. Test the client, get the weather.
5. Write a dynamic proxy client. You only need the generated interface type from the stub files.
6. Test the client, get the weather.
7. Write a DII client, none of the generated files are needed.
8. Test the client, get the weather.

Ex3-Web Services

RPC and Message Web Services

Exercise description:

In this exercise, two kinds of web services will be written and deployed:

1. An RPC-based web service with a user-defined parameter type (JavaBean)
2. A message-based web service (advanced)

To test the services, two clients will be used.

Exercise steps – RPC:

1. Write the RPC-based AddressService. This service will hold a list of addresses attached to user names and will have two methods:
   ```
   void addAddress(String userName, AddressBean address)
   AddressBean getAddress(String userName)
   ```
   This service must be stateful.
2. Deploy the service using Ant. Check the WSDL, see the new <types> element.
3. Write the client, first create the stub. Check the classes that were created and look for an additional class representing the AddressBean.
4. Add some users and addresses.

Exercise steps – message (Advanced):

1. Write the Message-based MessageService. This service will get an XML message sent to it and return another XML using a single method:
   ```
   Element[] echoElements(Element[] elems)
   ```
   The message to send is provided in the file message.xml, check it out.
   The result will look like this:
   ```
   <result>
       Message sent successfully to {destination here}.
   </result>
   ```

2. Deploy the service using Ant.
3. Write the client as a DII web service client.
4. Check the web service.

Ex4-SAAJ

Writing a SOAP client with SAAJ

Exercise description:

It this exercise, another client will be written for the WeatherService. This time, the client will be written with the SAAJ API, i.e., you will have to construct the SOAP request, send it and parse the SOAP response.
Remember that SAAJ is similar to DOM APIs, you must be familiar with the structure of a SOAP message to successfully complete this exercise.

Exercise steps:

1. We use the pre-deployed WeatherService, so no new service should be deployed.
2. Write the SAAJClient.
3. Run the client and see that you get the same response as before.

Ex5-WSDL

Top-Down Server Development

Exercise description:

In this exercise we will create a new web service, implementing an interface defined by a third body (e.g., an industry standard).

The interface is included in the provided **xr.wsdl** file. You have to create a web service that implements this WSDL, and deploy it.

Exercise steps:

1. Examine the included WSDL file, and identify the methods that your service should provide, and their parameter types and return types.
2. Use the wsdl2java Ant task, with the "serverside=true" setting, to create the server-side proxy.
3. Inspect the resulting interface file, ExchangeRates.java. Does it match the methods defined by the WSDL?
4. Implement this interface in a new Java class. (You can create a dummy implementation that will return random or preset values.)
5. Write a deployment descriptor (WSDD) for this class, and deploy it. (Note that Axis provides a skeleton for a WSDD file, but it is not sufficient.)

Ex6-JAXB

Object to XML mapping

Exercise description:

In this exercise we use the JAXB mechanism to unmarshal XML to Java objects, change the objects, validate the data and marshal it into a new XML document.
The JAXR APIs give you a great mechanism to work with MXL files instead of using the standard DOM parsing techniques or any other mechanism.

To work with JAXB, we need to use the JAXB compiler – xjc.
We will run it using an Ant build script.

Exercise steps:

1. An XML file (message.xml) and an XML schema file (message.xsd) are provided. Browse them and get to know them.
2. Run the xjc compiler using Ant. Check out the newly created classes.
3. Write the Main class, the following steps are needed:
    a. Unmarshal the message.xml file.
    b. Print the number of messages found in the file.
    c. Print the content of each message.
    d. Create a new message object (using the ObjectFactory generated by xjc).
    e. Set the subject of the new message to null. This is invalid according to the XML schema!
    f. Validate the message object. You will get an exception.
    g. Change the subject to something new.
    h. Marshal the object into a new XML file.
4. Browse the new file, see that the new message is added.

Ex7-Security

Authentication and Authorization with AXIS

Exercise description:

In this exercise we will add some security aspects to a Bank web service. The purpose is to authenticate a user before he accesses the service and then also check that he is authorized.

Here, we use a very simple text based user repository (users.lst) and two pre-existing AXIS handlers:

```
org.apache.axis.handlers.SimpleAuthenticationHandler
org.apache.axis.handlers.SimpleAuthorizationHandler
```

Exercise steps:

1. Define the username/password pair. Find the file named users.lst in the AXIS WEB-INF directory and add a new user and password.
2. Write the bank service. The bank service has a single method:
   `String getAccountBalance()`
   which returns the username and his account balance.
   To get the current username, use the `MessageContext` class.
3. Add the security handlers to the axis deployment file (deploy.wsdd), and deploy the service using Ant.
4. Write the Bank client as a DII client. We need to specify the username and password for the Call. Use the following methods:
   `Call.setProperty(String properyName,String value)`
5. Check the client with a proper user and with an unauthorized user. See what happens with an invalid user.

Ex8-J2EE Web Service

Writing a J2EE 1.4 EJB based Web Service

Exercise description:

In this exercise we will write a standard J2EE 1.4 web service based on a stateless session bean.
The web service will enable simple field validation for email fields and phone number fields.
If validation fails, a reason will be sent to the client.
On step 2, we will add a standard handler that logs all SOAP requests to the server.

The service will be packed as a standard ejb-jar file, and deployed to a standard J2EE 1.4 compatible application server.

Exercise steps:

1. The Validator stateless EJB already exists. Browse the bean sources and ejb-jar.xml deployment descriptor.
2. Implement the business methods.
3. To deploy the bean as a web service, we need to define a ValidatorEndpoint interface (should not be the same as the Validator object interface). This interface should include the methods that you want to allow access to from your web service client.
4. Generate the WSDL file from the endpoint interface using the Ant "createWSDL" target.
5. Write the webservices.xml DD file.
6. Browse the mapping.xml file. This file maps Java classes and packages to XML schema types.
7. Run the "deploy" target. **Make sure that the jar is deployed to the correct location.**
8. After deployment, open the browser and find the deployed service and WSDL.
9. Create the ValidatorClient. A "createStub" target is supplied.
10. Check your service.

Step 2

1. In this step, we will add a standard Handler to log our SOAP messages
2. Create the LogHandler class, this class extends the GenericHandler class. Add the missing methods.
3. Add the handler behavior, write the SOAPMessage to the standard output for every request.
4. Add the needed entry to the webservices.xml deployment descriptor.
5. Redeploy the service.
6. Run the client and test the handler.