Lab 10: Writing Message Handlers

In this lab, you will write a service that uses handlers.

Objectives:

- To write handlers

Part 1: Writing a Server-Side Handler

In this part, you will code and configure a handler that measures the elapsed time of requests to the Web service.

Steps:

_1. As you have done in earlier labs, create a new dynamic Web project named **lab10Web** associated with *lab10EAR*.

_2. In the lab10Web project, create a bottom-up "Hello, world" service. You can copy from the lab02Web project as necessary.

Deploy (add) lab10EAR to the server and ensure that it's published.

Test the service in the Web Services Explorer to ensure that it works.

_3. To ensure that there's a measurable elapsed time for the handler you will write to measure, edit the service method and add the following (before the *return*):

```
try
{
    Thread.sleep(100);
}
catch(InterruptedException e) {}
```

_4. In a new package named **handlers**, create a class named **LoggingHandler** that implements the *SOAPHandler<SOAPMessageContext>* interface. This class will need the following methods (hint: use the *Source - Override/Implement Methods* menu after you create the class):

- close
- handleFault
- handleMessage
- getHeaders

The handleFault and handleMessage methods should return *true* and getHeaders should return *null*.

Ensure that you have no errors at this point.

_5. For debugging purposes, put an informative *System.out.println* in each of the overriden methods, for example:

```
System.out.println("In handleFault");
```

_6. Complete the *handleMessage* method:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

a. Rename the arg0 parameter to **ctx**.

b. Write an *if* statement that determines if the message is an inbound (request) or outbound (response) message.

c. If it's an inbound (request) message, simply use System.out.println to write a message to the console indicating the request was received.

_7. In the same package where the service implementation resides (e.g. *service*), create a file named **handler-chain.xml**.

_8. Complete the XML file so it looks like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<javaee:handler-chains
    xmlns:javaee="http://java.sun.com/xml/ns/javaee"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <javaee:handler-chain>
  <javaee:handler>
   <javaee:handler-class>
     handlers.LoggingHandler
   </javaee:handler-class>
  </javaee:handler>
 </javaee:handler-chain>
</javaee:handler-chains>
```

_9. In your service implementation class, add the following annotation above the class definition:

```
@HandlerChain(file = "handler-chain.xml")
```

_10. Republish the EAR, then as before, test with the Web Services Explorer. You should see your handler's message in the RAD console.

_11. Now update the handler so it measures the service's performance. Modify the *handleMessage* method:

a. If the message is a request message, note the current time and store it in the message context:

```
long startTime = System.currentTimeMillis();
ctx.put("startTime", startTime);
```

b. If the message is a response message, note the new current time, retrieve the start time from the context, and calculate and display the elapsed time:

```
long endTime = System.currentTimeMillis();
long startTime = (Long)ctx.get("startTime");
System.out.println("Elapsed time: " + (endTime - startTime) +
  " milliseconds");
```

c. Republish and test in the WSE - you should see the elapsed time in the RAD console.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Part 2: Logging More Information

In this part, you will use the SAAJ API in your request handler to log the credit-card number being validated.

Steps:

_1. Using the SAAJ chapter in the book and the SAAJ JavaDocs, modify your request handler so that it prints the request message's *name* element's value to the console.

Hint: When you iterate elements using SAAJ, you need to filter out spurious Text nodes and find SOAPElement nodes. You can use code like:

```
SOAPElement payload = null;
Iterator iter = body.getChildElements();
while (iter.hasNext())
{
    org.w3c.dom.Node n = (org.w3c.dom.Node)iter.next();
    if (n instanceof SOAPElement)
    {
        payload = (SOAPElement)n;
    }
}
```

Part 3: Working with Headers

In this part, you will write client and server-side handlers. The client-side handler will insert a header into the SOAP message and the server-side handler will extract and process the header.

Our service recognizes the following request header:

```
<bc:BigCompanyRequestHeader
    xmlns:bc="http://service.bigcompany.com">
  <app-version>version number (xsd:int)</app-version>
  <client-email>email address (xsd:string)</client-email>
</bc:BigCompanyRequestHeader>
```

The service generates the following response header:

```
<bc:BigCompanyResponseHeader
     xmlns:bc="http://service.bigcompany.com">
  <request-id>request identifier (xsd:int)</request-id>
  <response-time>response time in milliseconds(xsd:int)</response-time>
</bc:BigCompanyResponseHeader>
```

Steps:

_1. Create a new Dynamic Web project named **lab10Part3Web** associated with *lab10Part3EAR*.

_2. Create a new folder named **wsdl** in the project's WEB-INF folder.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

_3. Perform a File System import of the **{Lab Installation Directory}/starters/lab08/InvoiceService.wsdl** file into the *wsdl* directory.

_4. Open the WSDL, and note the following:

a. Within the *types* section, within the existing schema, there are elements named **BigCompanyRequestHeader BigCompanyResponseHeader** that validate the header XML shown above.

b. There are *message* elements that use the elements defined in the schema:

```
<wsdl:message name="requestHeaderMessage">
  <wsdl:part name="requestHeader"
     element="intf:BigCompanyRequestHeader" />
</wsdl:message>

<wsdl:message name="responseHeaderMessage">
  <wsdl:part name="responseHeader"
     element="intf:BigCompanyResponseHeader" />
</wsdl:message>
```

c. In the *binding*, there are *header* elements defined for the both the request and response messages, for example:

```
<wsdlsoap:header use="literal" part="requestHeader"
    message="intf:requestHeaderMessage"/>
```

_5. As you have done in earlier labs, generate a top-down Web service from the WSDL into the lab10Part3 package.

Complete the service method so that it returns *true*.

_6. Test the service to ensure that it works so far:

a. Right-click on InvoiceService.wsdl and choose *Web Services - Test with Web Services Explorer*. **Note:** Be sure to test with the WSDL, NOT with the *Services* entry in the Enterprise Explorer.

b. Click the *payInvoice* link, enter reasonable data for the header and payload elements, then invoke the service and ensure there's no error and the service returns *true*.

c. Examine the SOAP message source and ensure that the request message has the correct header. The response message will not as yet have any headers.

_7. Now let's work on the server-side handler that will process the header. In the *lab10Part3Web* project, create a new class named **ServerHeaderHandler** in the *handlers* package that extends *SOAPHandler<SOAPMessageContext>*.

_8. Complete the handler:

a. Modify the *getHeaders* method so that it looks like:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
public Set<QName> getHeaders()
{
  Set<QName> headers = new HashSet<QName>();
  headers.add(new QName("http://service.bigcompany.com",
        "BigCompanyRequestHeader"));

  return headers;
}
```

b. In the *handleMessage* method, if it's a request (inbound) message use SAAJ to access the *app-version* and *client-email* values and display them to the console.

   Ensure that the handleMessage method returns *true*.

_9. In the same fashion as before, create a handler-chain XML file and configure the handler on the service endpoint.

_10. Test in the WSE as before and look in the RAD console for the header values.

_11. If you're feeling ambitious, complete the application:

- Modify the server-side handler so it inserts the BigCompanyResponseHeader described at the top of this part into response messages. For the *request-id*, generate a random integer.
- Write a servlet-based client with a client-side handler that inserts the BigCompanyRequestHeader and processes the BigCompanyResponseHeader.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.