Implementing SOAP Services by Using JAX-WS

6

ORACLE

Objectives

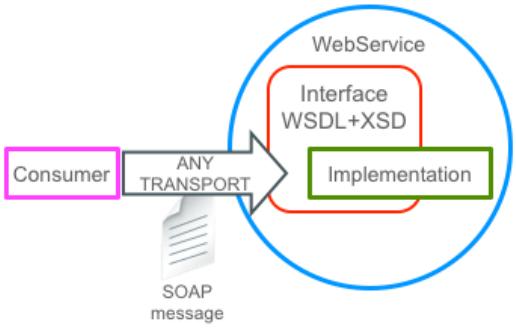
After completing this lesson, you should be able to handle communications with SOAP Services, including how to:

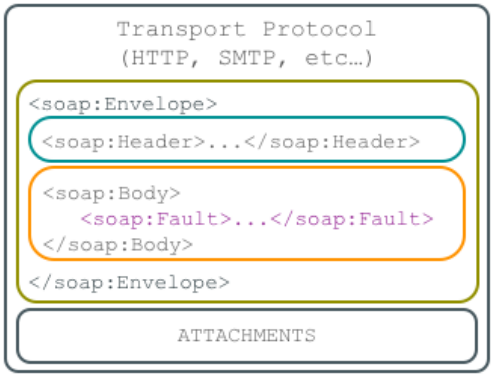
- Describe a SOAP Web Service structure
- Create SOAP Web Services using JAX-WS API
- Create SOAP Web Service clients

WebServices and SOAP

SOAP WebServices

- Web Services are interoperable, platform independent mechanism for component interactions.
- SOAP decouples message representation from transport.
- Service Interface is described with WSDL and XSD files.
- Service Implementation are disguised — Consumer is unaware of implementation details of the service.





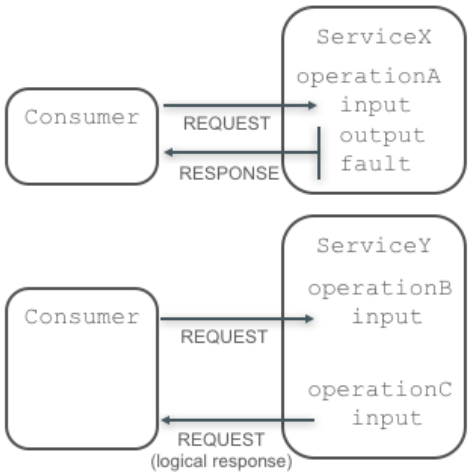
SOAP Message

- Can be transported over different protocols
- Must be enclosed into SOAP Envelope
- May have Headers, such as security, reliability, and so on
- Must have a Body with business data, or Fault elements
- May have attachments

Web Service Interaction Patterns

Web Services may use different interaction patterns.

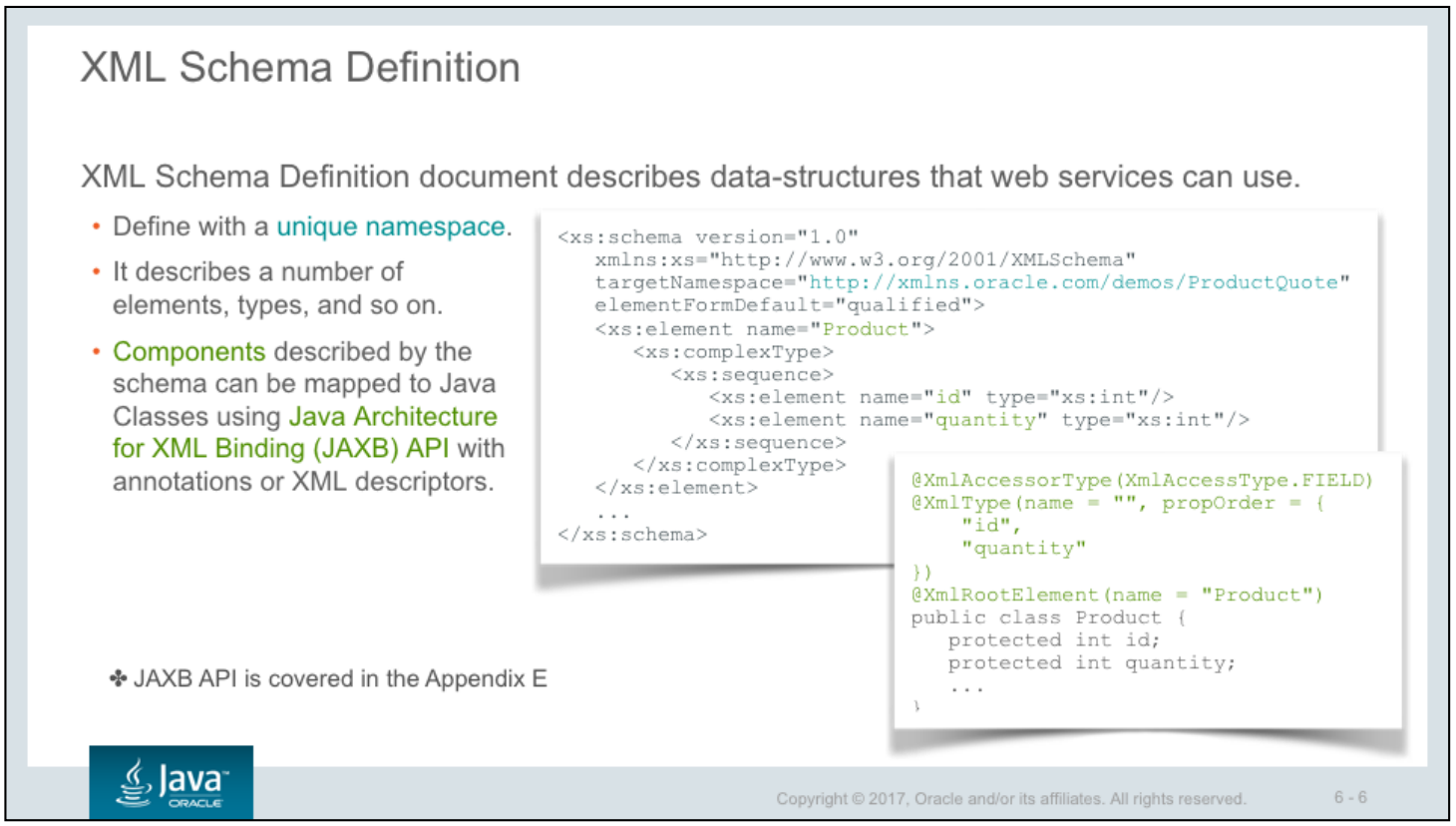
- Synchronous Service Operations:
  - Describe Input, Output and optionally Fault parts
  - Consumer that invokes such operation sends a request containing the input and expects either fault to output response to be returned from the service.
  - Consumer is blocked for the duration of the call.
- Asynchronous Service Operations:
  - One-Way operations accept request from consumer and do not produce any response.
  - Two-Way operations accept request from consumer and may perform callback sending another request back to the original consumer.
  - Two-Way SOAP operations can be coordinated with WS-Addressing standard that describes callback addresses and correlations.



Web Service Interface

Web Services Description Language (WSDL) is an XML-based standard for describing SOAP Web Services interfaces.

- XML Schema Definition (XSD) documents describe data structures that services can use.
- WSDL file describes the service with:
  - Abstract WSDL part:
    - References XSD file to use data-structures it defines
    - Describes messages comprising elements from the XSD that this service would accept and return in its operations
    - Describes port types - logical groups of operations
    - Describes operations that would use messages to produce input, output, or fault parts
  - Concrete WSDL part:
    - Describes bindings to define how to invoke operations
    - Describes service to define an end-point invocation address where this service can be found



An example of the XML Schema definition file that models data structures for future use of the web service request, response, and fault messages:

```
<xs:schema version="1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://xmlns.oracle.com/demos/ProductQuote"
           elementFormDefault="qualified">
    <xs:element name="Product">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="id" type="xs:int"/>
                <xs:element name="quantity" type="xs:int"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="Quote" type="xs:float"/>
    <xs:element name="QuoteError" type="xs:string"/>
</xs:schema>
```

Java class that is mapped to this xml schema Product element using JAXB annotations:

```
package com.oracle.xmlns.demos.productquote;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "id",
    "quantity"
})
@XmlRootElement(name = "Product")
public class Product {
    protected int id;
    protected int quantity;
    public int getId() {
        return id;
    }
    public void setId(int value) {
        this.id = value;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int value) {
        this.quantity = value;
    }
}
```

WSDL Schemas and Namespaces

Web Services Description Language (WSDL) namespace definitions

- References XSD via its unique namespace to access data structures described by the XML Schema
- Defines its own unique namespace, for referencing components described within this WSDL file
- Each namespace can be assigned an arbitrary prefix tns ns1 and so on

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PriceQuote" targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService"
             xmlns:tns="http://xmlns.oracle.com/demos/PriceQuoteService"
             xmlns:ns1="http://xmlns.oracle.com/demos/ProductQuote"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
       <xsd:schema>
           <xsd:import namespace="http://xmlns.oracle.com/demos/ProductQuote"
                       schemaLocation="ProductQuote.xsd"/>
       </xsd:schema>
   </types>
   ...
</definitions>
```

WSDL Messages, PortTypes, and Operations

Abstract WSDL describes capabilities of the service:

- Messages, which comprise parts referencing elements described by XSDs
- Port Types, which define logical groups of operations
- Operations, which define:
  - input
  - output
  - fault

  referencing previously defined messages

```
<?xml version="1.0"?>
<definitions ...>
  <types>...</types>
  <message name="QuoteRequest">
      <part name="request" element="ns1:Product"/>
  </message>
  <message name="QuoteResponse">
      <part name="response" element="ns1:Quote"/>
  </message>
  <message name="QuoteFault">
      <part name="response" element="ns1:QuoteError"/>
  </message>
  <portType name="PriceQuote">
       <operation name="getQuote">
            <input message="tns:QuoteRequest"/>
            <output message="tns:QuoteResponse"/>
            <fault name="QuoteFault"
                   message="tns:QuoteFault"/>
       </operation>
  </portType>
```

Here is an example of the Web Service Description file that models data structures for future use of the web service request, response, and fault messages:

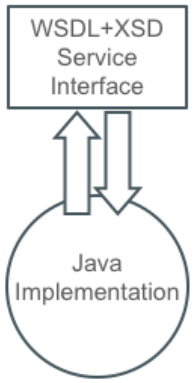
```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PriceQuote"
targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService"
             xmlns:tns="http://xmlns.oracle.com/demos/PriceQuoteService"
             xmlns:ns1="http://xmlns.oracle.com/demos/ProductQuote"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <xsd:schema>
            <xsd:import
namespace="http://xmlns.oracle.com/demos/ProductQuote"
                        schemaLocation="ProductQuote.xsd"/>
        </xsd:schema>
    </types>
    <message name="QuoteRequest">
        <part name="request" element="ns1:Product"/>
    </message>
```

```
  <message name="QuoteResponse">
          <part name="response" element="ns1:Quote"/>
      </message>
      <message name="QuoteFault">
          <part name="response" element="ns1:QuoteError"/>
      </message>
      <portType name="PriceQuote">
          <operation name="getQuote">
              <input message="tns:QuoteRequest"/>
              <output message="tns:QuoteResponse"/>
              <fault name="QuoteFault" message="tns:QuoteFault"/>
          </operation>
      </portType>
      <binding name="PriceQuoteSOAP" type="tns:PriceQuote">
          <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
          <operation name="getQuote">
              <input>
                  <soap:body use="literal" parts="request"/>
              </input>
              <output>
                  <soap:body use="literal" parts="response"/>
              </output>
              <fault name="QuoteFault">
                  <soap:fault name="QuoteFault" use="literal"/>
              </fault>
          </operation>
      </binding>
      <service name="PriceQuote">
          <port name="PriceQuote" binding="tns:PriceQuoteSOAP">
              <soap:address
location="http://localhost:7001/demos/PriceQuote"/>
          </port>
      </service>
  </definitions>
```

Top-down versus Bottom-up approach

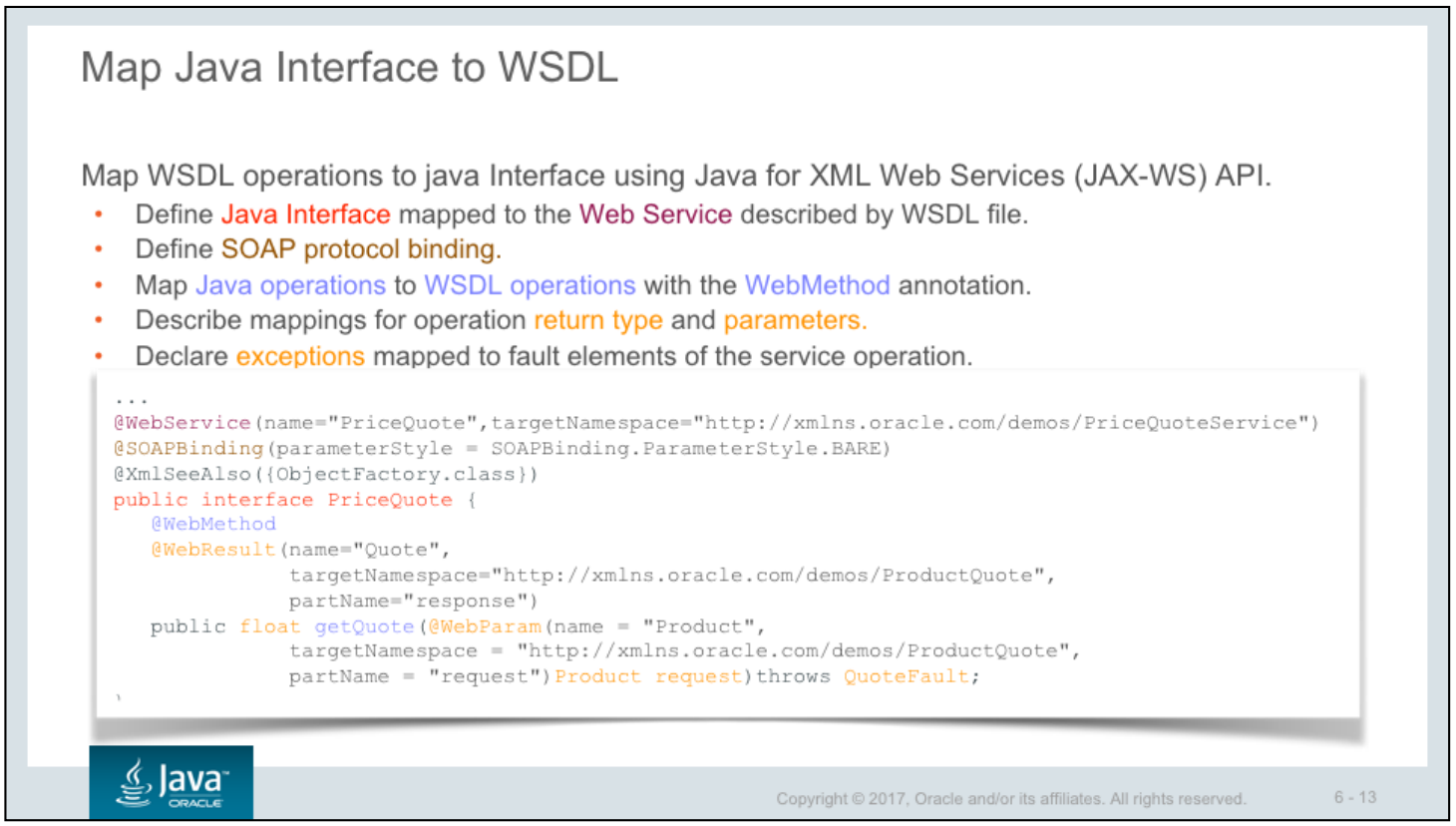
Top-down approach (contract first)

- Start Web Service development by defining service interface using WSDL and XSD files
- Then create Java classes mapped to this interface definition
- Java service implementation classes can be auto-generated by the IDE
- Provides better quality Web Service interface description



Bottom-up approach (implementation first)

- Start Web Service development by defining Java classes that will implement the service
- Then create Web Service interface definition artefacts (WSDL and XSD files)
- WebService interface definition files can be auto-generated upon deployment
- Quicker development



JAX-WS Requirements for Web Service Methods

- Must be public. By default, every public method in the class will be a part of the web service.
- Must not be static or final
- Must have JAXB-compatible parameters and return types. Parameters and return types must not implement the java.rmi.Remote interface.

WebMethod annotations used to map Java method to WSDL operation. It has following properties:

- operationName: Name of the wsdl:operation matching this method
- exclude: Marks a method to NOT be exposed as a web method
- action: Associate this operation with the SOAP Action property. Unique label for this operation, that may optionally be passed as a SOAP Header to link client call to specific operation.

The SOAPBinding annotation defines parameter style mapping for the web service. It could have two values:

- BARE: Method parameters represent the entire message body.
- WRAPPED: Parameters are elements wrapped inside a top-level element named after the operation.

The WebResult annotation defines the datatype mapping between the return type of the Java operation and output message used by the operation described within the wsdl file.

The WebParam annotation defines mapping between parameters of the Java Method and input message used by the operation described within the wsdl file.

XMLSeeAlso annotation instructs JAXB API to bind other classes (in this case ObjectFactory.class) when binding this (PriceQuote.class).

ObjectFactory class contains helper methods to produce java objects (Product, Quote, QuoteFault) mapped to input, output, and fault elements used by operation getQuote that this interface maps for the service implementation.

Here is a complete interface code example:

```
  package com.oracle.xmlns.demos.pricequoteservice;
  import javax.jws.WebMethod;
  import javax.jws.WebParam;
  import javax.jws.WebResult;
  import javax.jws.WebService;
  import javax.jws.soap.SOAPBinding;
  import javax.xml.bind.annotation.XmlSeeAlso;
  import com.oracle.xmlns.demos.productquote.ObjectFactory;
  import com.oracle.xmlns.demos.productquote.Product;
  @WebService(name = "PriceQuote", targetNamespace =
"http://xmlns.oracle.com/demos/PriceQuoteService")
  @SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
  @XmlSeeAlso({
        ObjectFactory.class
    })
    public interface PriceQuote {
       @WebMethod
        @WebResult(name = "Quote",
                   targetNamespace =
"http://xmlns.oracle.com/demos/ProductQuote",
                   partName = "response")
        public float getQuote(
            @WebParam(name = "Product",
                      targetNamespace =
"http://xmlns.oracle.com/demos/ProductQuote",
                      partName = "request")
            Product request) throws QuoteFault;
    }
```

JAX-WS Implementation

Create Java implementations of SOAP Services using JAX-WS API.

- Define Java class that will provide Web Service implementation described by the WSDL file.
- Reference previously described Java interface that mapped service operations.
- Describe protocol Binding Type (in this case, SOAP1.2 over HTTP).
- Define Web Methods Implementations for operations declared by the Web Service Java Interface.

```
...
@WebService(serviceName="PriceQuote",
            portName="PriceQuote",
            endpointInterface="com.oracle.xmlns.demos.pricequoteservice.PriceQuote",
            targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService",
            wsdlLocation="WEB-INF/wsdl/PriceQuote/PriceQuoteService.wsdl")
@BindingType(value="http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP/")
public class PriceQuote {
   public float getQuote(Product request) throws QuoteFault {
      ...
   }
`
```

The BindingType annotation is used to specify the binding to use for a web service endpoint implementation class. Default is SOAP1.1/HTTP. This example uses SOAP1.2/HTTP binding type.

```
    package demos.ws;
    import com.oracle.xmlns.demos.pricequoteservice.QuoteFault;
    import com.oracle.xmlns.demos.productquote.Product;
    import javax.jws.WebService;
    import javax.xml.ws.BindingType;
    @WebService(serviceName = "PriceQuote",
                portName = "PriceQuote",
                endpointInterface =
"com.oracle.xmlns.demos.pricequoteservice.PriceQuote",
                targetNamespace =
"http://xmlns.oracle.com/demos/PriceQuoteService",
                wsdlLocation = "WEB-
INF/wsdl/PriceQuote/PriceQuoteService.wsdl")
    @BindingType(value =
"http://java.sun.com/xml/ns/jaxws/2003/05/soap/bindings/HTTP/")
    public class PriceQuote {
        public float getQuote(Product request) throws QuoteFault {
            ...
        }
    }
```

Create Java JAX-WS Client

JAX-WS API is used to implement both SOAP WebServices and Clients.

- Define Java class that will represent Web Service Client.
- Define Web Service Port Type references.

```
...
@WebServiceClient(name="PriceQuote",
                  targetNamespace="http://xmlns.oracle.com/demos/PriceQuoteService",
                  wsdlLocation="file:wsdl/PriceQuoteService.wsdl")
public class PriceQuote_Service extends Service {
   ...
   @WebEndpoint(name="PriceQuote")
   public PriceQuote getPriceQuote() {
        return super.getPort(
               new QName("http://xmlns.oracle.com/demos/PriceQuoteService", "PriceQuote"),
                         PriceQuote.class);
    }
}
```


Complete example of a Java client class that represents PriceQuote Web Service:

```
package com.oracle.xmlns.demos.pricequoteservice;
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 = "PriceQuote", targetNamespace =
"http://xmlns.oracle.com/demos/PriceQuoteService", wsdlLocation =
"file:wsdl/PriceQuoteService.wsdl")
public class PriceQuote_Service extends Service {
    private final static URL PRICEQUOTE_WSDL_LOCATION;
    private final static WebServiceException PRICEQUOTE_EXCEPTION;
    private final static QName PRICEQUOTE_QNAME = new
```

```
  QName("http://xmlns.oracle.com/demos/PriceQuoteService", "PriceQuote");
      static {
          URL url = null;
          WebServiceException e = null;
          try {
              url = new URL("file:wsdl/PriceQuoteService.wsdl");
          } catch (MalformedURLException ex) {
              e = new WebServiceException(ex);
          }
          PRICEQUOTE_WSDL_LOCATION = url;
          PRICEQUOTE_EXCEPTION = e;
      }
      public PriceQuote_Service() {
          super(__getWsdlLocation(), PRICEQUOTE_QNAME);
      }
      public PriceQuote_Service(WebServiceFeature... features) {
          super(__getWsdlLocation(), PRICEQUOTE_QNAME, features);
      }
      public PriceQuote_Service(URL wsdlLocation) {
          super(wsdlLocation, PRICEQUOTE_QNAME);
      }
      public PriceQuote_Service(URL wsdlLocation, WebServiceFeature...
features) {
          super(wsdlLocation, PRICEQUOTE_QNAME, features);
      }
      public PriceQuote_Service(URL wsdlLocation, QName serviceName) {
          super(wsdlLocation, serviceName);
      }
      public PriceQuote_Service(URL wsdlLocation, QName serviceName,
WebServiceFeature... features) {
          super(wsdlLocation, serviceName, features);
      }
      @WebEndpoint(name = "PriceQuote")
      public PriceQuote getPriceQuote() {
          return super.getPort(new
QName("http://xmlns.oracle.com/demos/PriceQuoteService", "PriceQuote"),
PriceQuote.class);
      }
```

```
  @WebEndpoint(name = "PriceQuote")
      public PriceQuote getPriceQuote(WebServiceFeature... features) {
          return super.getPort(new
QName("http://xmlns.oracle.com/demos/PriceQuoteService", "PriceQuote"),
PriceQuote.class, features);
      }
      private static URL __getWsdlLocation() {
          if (PRICEQUOTE_EXCEPTION!= null) {
              throw PRICEQUOTE_EXCEPTION;
          }
          return PRICEQUOTE_WSDL_LOCATION;
      }
  }
```

Invoke SOAP Service from Java SE Client

Invoke Web Service from Java SE client.

- Acquire Web Service Port.
- Prepare input parameters.
- Invoke service operations.
- Handle output values and possible faults.

```
...
public static void main(String[] args) {
   PriceQuote priceQuotePort = new PriceQuote_Service().getPriceQuote();
   Product product = new Product();
   product.setId(1);
   product.setQuantity(4);
   try {
      float quote = priceQuotePort.getQuote(product);
   }catch(QuoteFault ex) {...}
}
...
```

Invoke SOAP Service from Java EE Component

Invoke Web Service from Java EE Container Managed component such as Servlet or EJB.

- Inject Web Service Reference and acquire Web Service Port.
- Prepare input parameters.
- Invoke service operations.
- Handle output values.
- Handle possible faults.

```
...
@Stateless
public class OrderManagement {
   @WebServiceRef(
   wsdlLocation="http://localhost:7001/demos/PriceQuote?WSDL")
   private PriceQuote_Service service;
   public float getQuote(int id, int quantity)
                                        throws OrderException{
      PriceQuote priceQuotePort = service.getPriceQuote();
      Product product = new Product();
      product.setId(1);
      product.setQuantity(4);
      try {
         float quote = priceQuotePort.getQuote(product);
         return quote;
      }catch(QuoteFault ex) {...}
   }
}
```


When handling web service operation input, output, and fault elements, consider converting JAXB object provided by the web service into your application objects.

Summary

In this lesson, you should have learned how to handle communications with SOAP Services, including how to:

- Describe a SOAP Web Service structure
- Create SOAP Web Services using JAX-WS API
- Create SOAP Web Service clients

Practice Overview

This practice covers the following tasks:

- Create Price Quote Service Based on existing WSDL and XSD files
- Create Java Client Application that consumes the Price Quote Service