9 Top-Down JAX-WS Web Services

Objectives

After completing this lesson, you should be able to do the following:

- Describe the benefits of WSDL First Design
- Generate Service Endpoint Interfaces (SEIs) from WSDLs
- Implement Service Endpoint Interfaces
- Customize SEI Generation

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 1: Introduction to Web Services
- Lesson 2: Creating XML Documents
- Lesson 3: Processing XML with JAXB
- Lesson 4: SOAP Web Services Overview
- Lesson 5: Creating JAX-WS Clients

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 6: Exploring REST Services
- Lesson 7: Creating REST Clients
- Lesson 8: Bottom Up JAX Web Services
- **Lesson 9: Top Down JAX Web Services** — You are here!
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

- Lesson 11: Web Service Error Handling
- Lesson 12: Java EE Security and Securing JAX WS

JAX-WS Development Approaches

- Generate web service artifacts using the information contained in a WSDL file.
- Create service endpoint interface (SEI) or value classes as Java source files, and then use them as inputs to generate the associated WSDL descriptor and other portable artifacts.

In either case, JAX-WS generates the majority of the infrastructure code required by the service.

Strong Typing for Web Services:

A Benefit of the Top-Down Approach

Fully worked WSDL can specify data and message types to fine granularity, using the full power of XML Schema and allowing clients as well as servers to validate message content locally as follows:

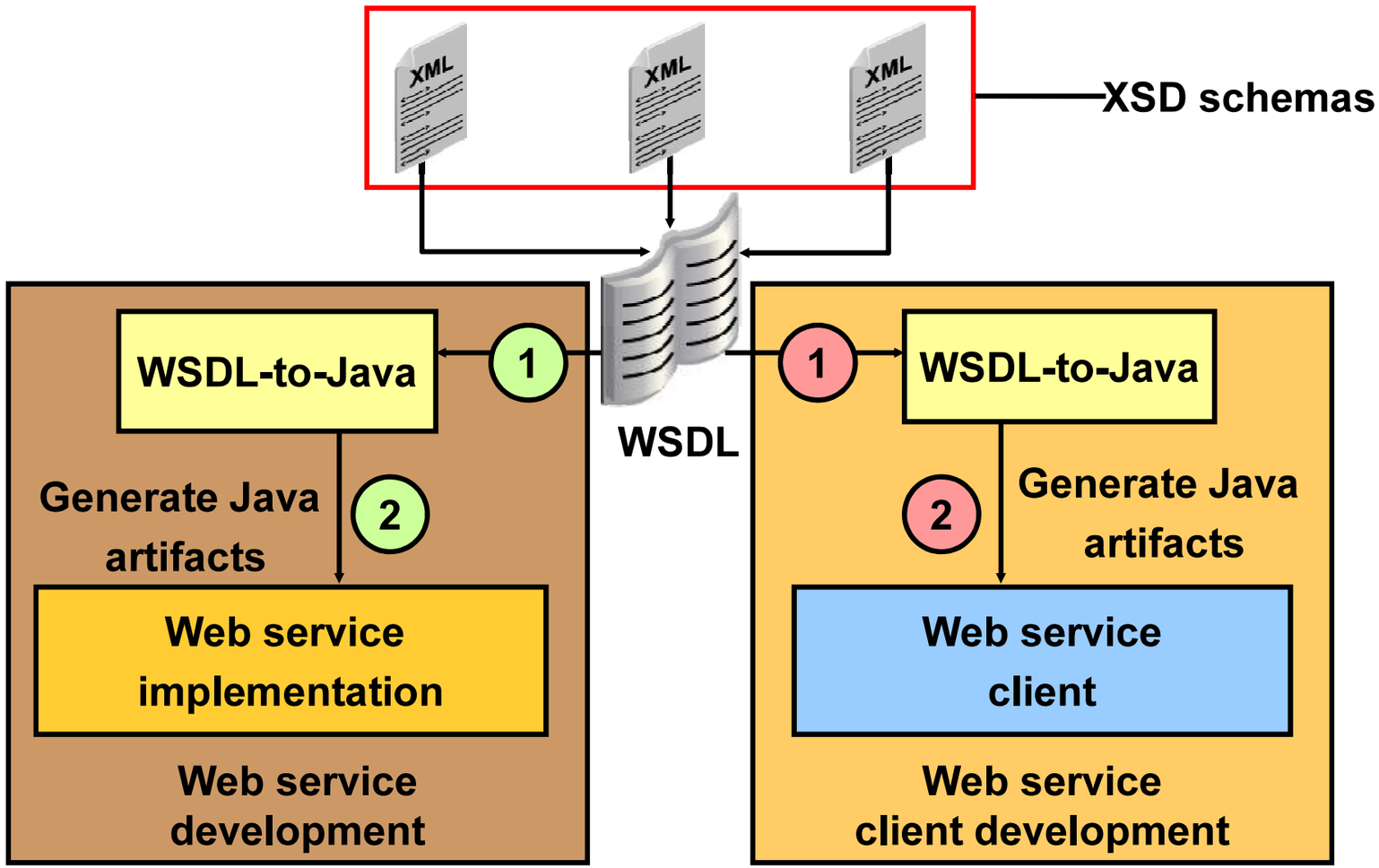
- When you generate Java from WSDL, you can map strong XML types to Java and validate those types using generated code.
- When you generate WSDL from Java, the generated types are weaker.

Benefits of a Top-Down Approach

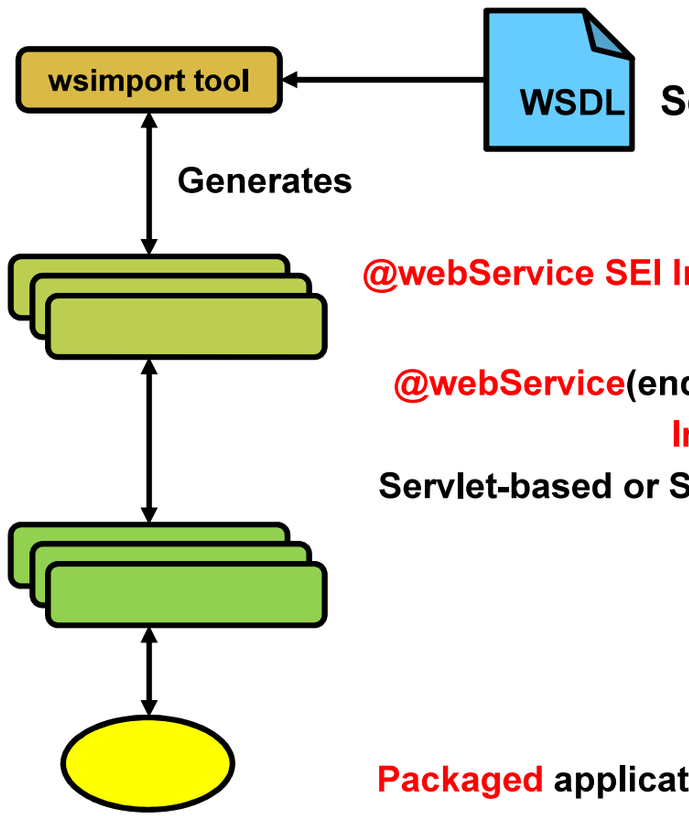
Working from WSDL provides the following advantages:

- WSDL is better suited when the service developer is not also the author of the service.
- The strong typing is shared using the WSDL file itself.
- Interoperability is easier to achieve.
- Serializable types are expressed first in the XML Schema and then mapped to one or more program languages and object models.
- Client-side validation is easier to implement.

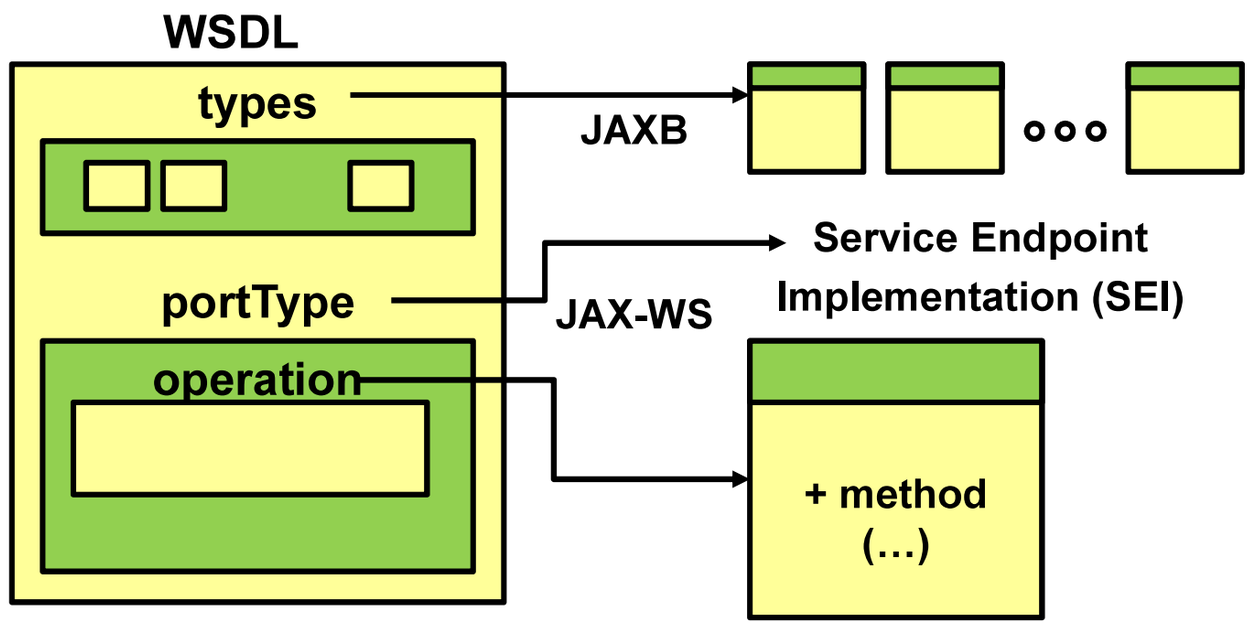
Top-Down Approach



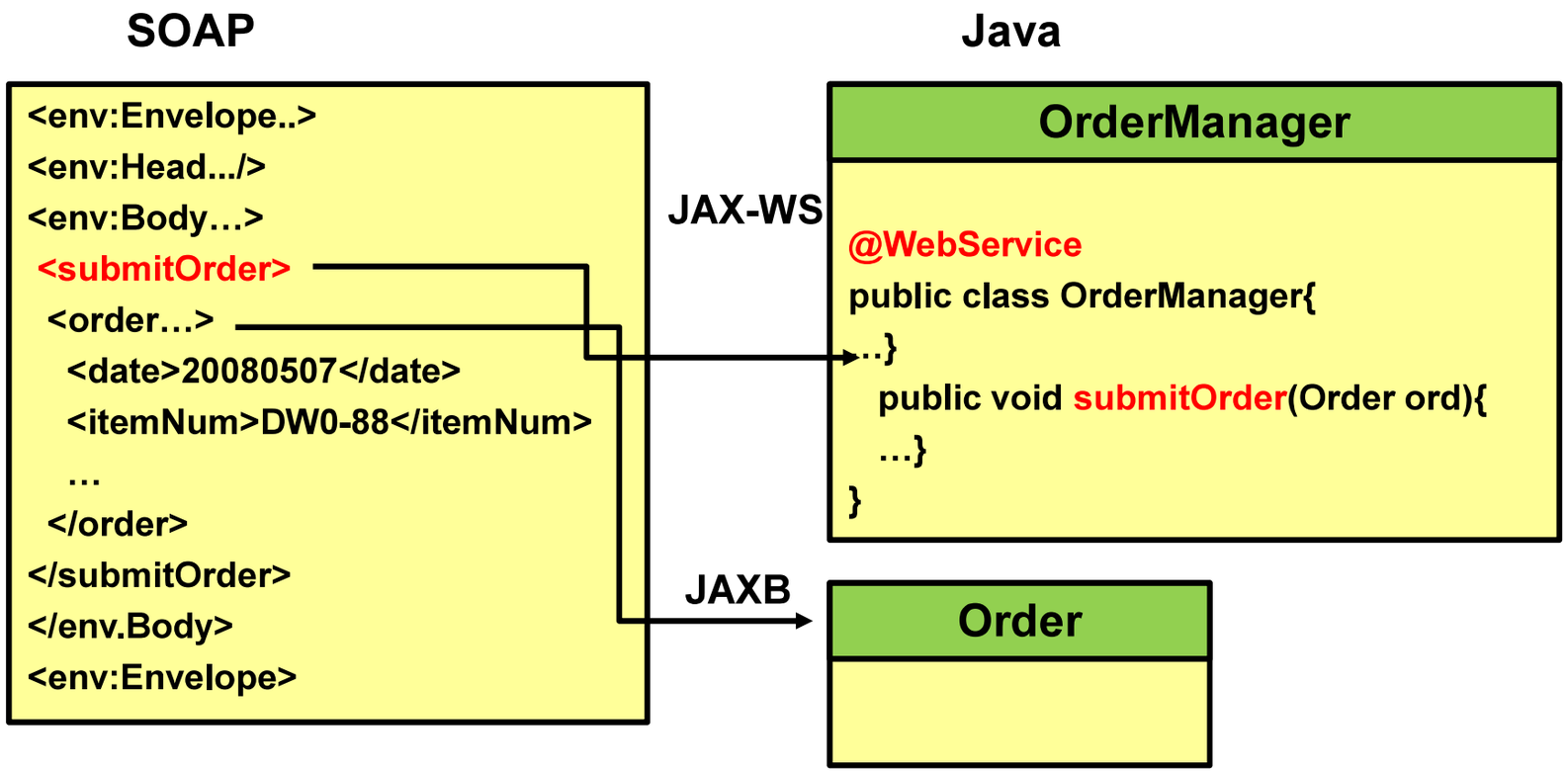
Starting from a WSDL Description



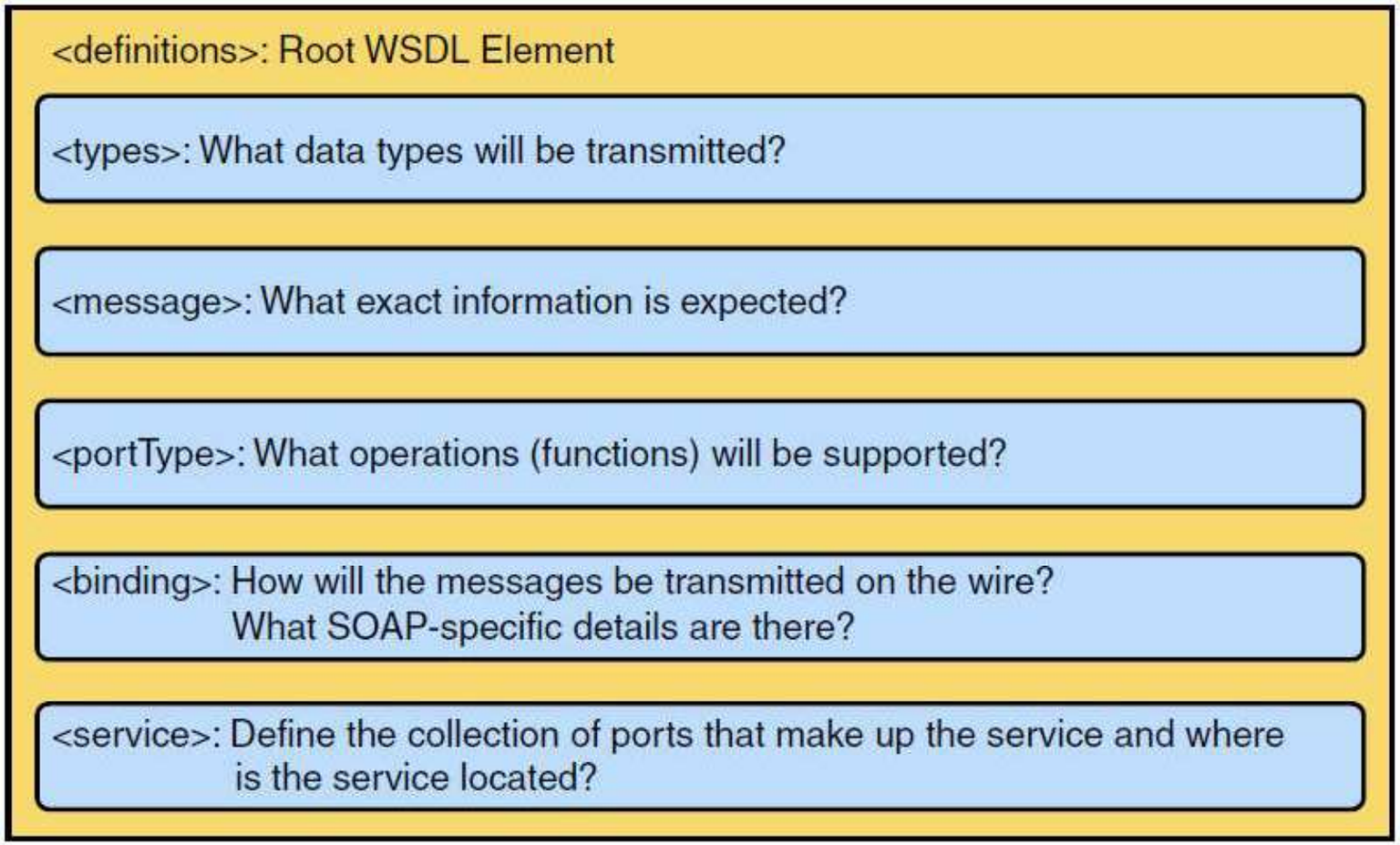
JAX-WS:
WSDL to Java



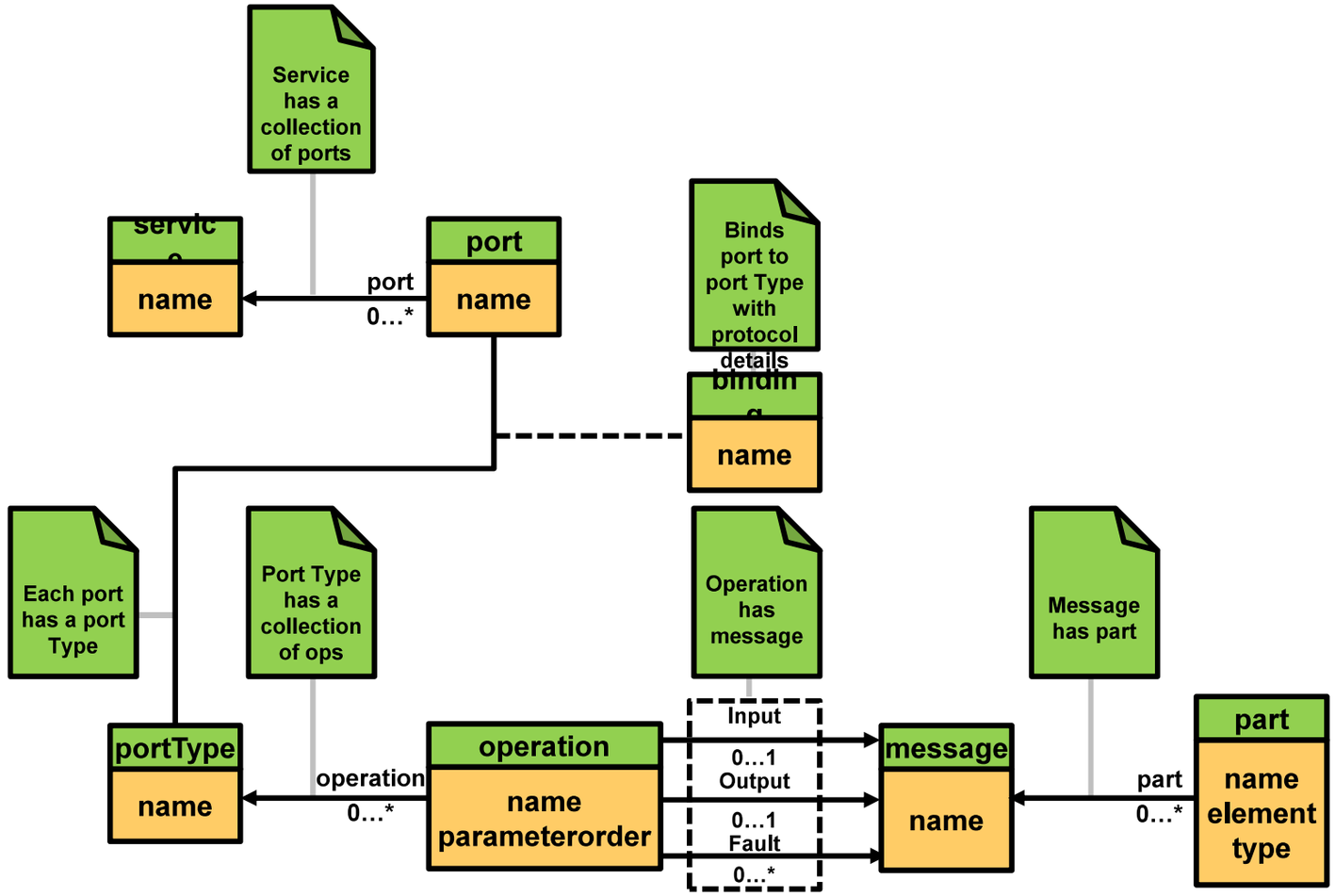
JAX-WS:
SOAP to Java



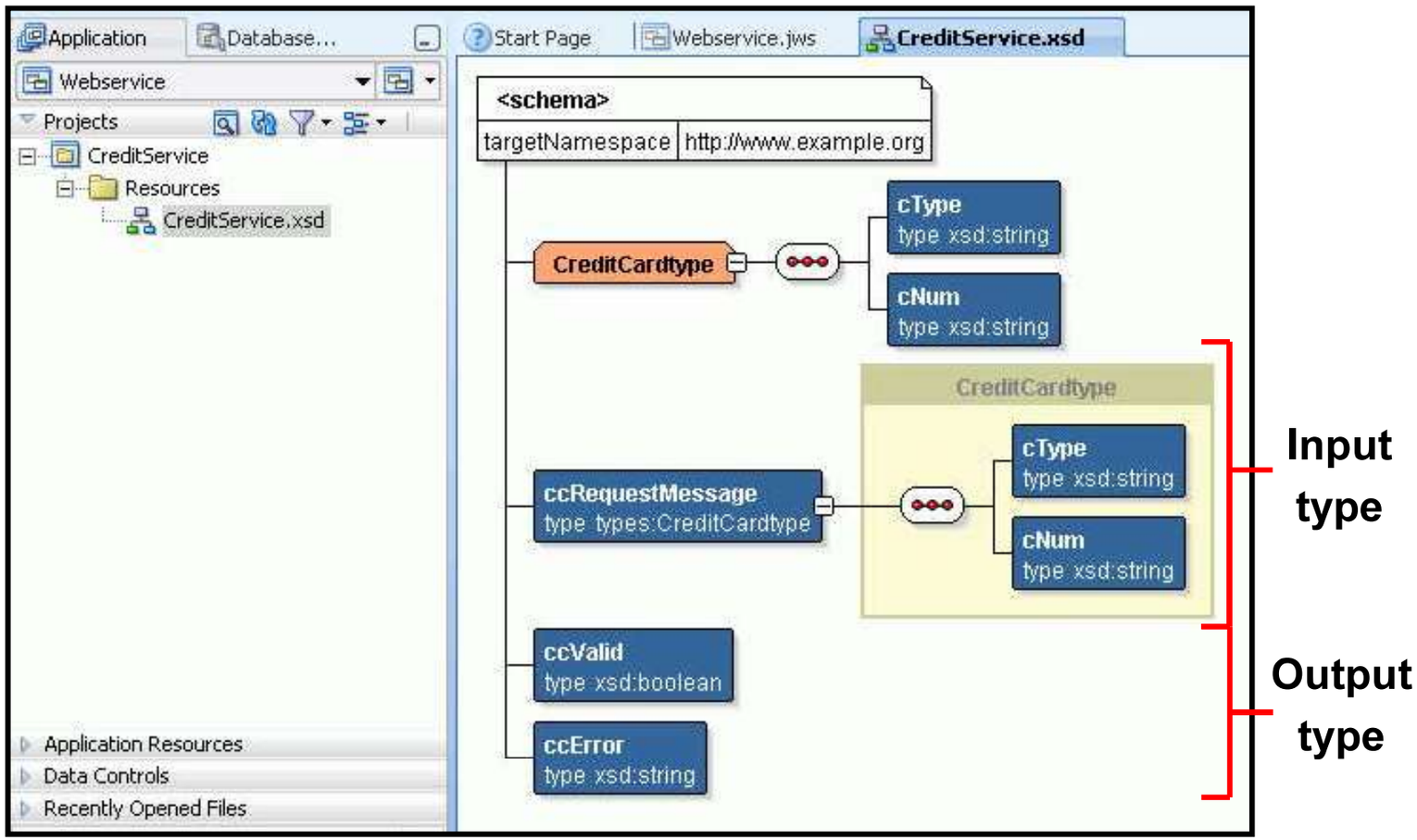
Structure of a WSDL file



Definition of a Service Using WSDL



Defining an XSD Schema



Creating WSDL:
WSDL Description (1 of 2)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="PassengerManagerPort.wsdl"
3       xmlns="http://schemas.xmlsoap.org/wsdl/"
4       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7       xmlns:tns="urn://Traveller/"
8       targetNamespace="urn://Traveller/">
9    <types>
10      <xsd:schema>
11        <xsd:import namespace="urn://Traveller/"
12           schemaLocation="PassengerManagerSchema.xsd"/>
13      </xsd:schema>
14 </types>
```

Creating WSDL:
WSDL Description (2 of 2)

```
15  <message name="addPassengerRequest">
16   <part name="params" element="tns:addPassenger"/>
17  </message>
18  <message name="addPassengerResp">
19   <part name="params" element="tns:addPassengerResponse"/>
20  </message>
21  <portType name="PassengerManager">
22   <operation name="addPassenger">
23     <input name="in1" message="tns:addPassengerRequest"/>
24     <output name="out1" message="tns:addPassengerResp"/>
25    </operation>
26  </portType>
27 </definitions>
```

Creating WSDL:
WSDL Schema

```
1  <xsd:schema id="PassengerManagerSchema.xsd"
2       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3       xmlns:tns="urn://Traveller/"
4       elementFormDefault="qualified"
5       targetNamespace="urn://Traveller/">
6    <xsd:element name="addPassenger">
7       <xsd:complexType>
8        <xsd:sequence>
9         <xsd:element name="firstName" type="xsd:string"/>
10        <xsd:element name="lastName" type="xsd:string"/>
11       </xsd:sequence>
12      </xsd:complexType>
13 </xsd:element>
```

Creating WSDL:
WSDL Bindings

```
1  <binding name="binding" type="tns:PassengerManager">
2    <soap:binding style="document"
3      transport="http://schemas.xmlsoap.org/soap/http"/>
4    <operation name="addPassenger">
5      <soap:operation/>
6      <input><soap:body use="literal"/></input>
7      <output><soap:body use="literal"/></output>
8    </operation>
9  </binding>
10 <service name="PassengerManagerService">
11   <port name="PassengerManager" binding="tns:binding">
12     <soap:address
13       location="http://localhost:8080/passengerManager"/>
14   </port>
15 </service>
```

`wsimport` Generated Artifacts

Running the `wsimport` command generates all required JAX-WS artifacts. These artifacts can be categorized as:

- JAXB Artifacts
  - Each complexType from (or included in) the types section of the WSDL will have a generated Java class.
  - Each input, output, and fault message for each operation will have a JAXB class.
    - An operation will have a GetGroupDescription.java and GetGroupDescriptionResponse.java for a get-group-description operation.
  - ObjectFactory.java
  - package-info.java
- *ServiceName*.java – Used by clients
- *PortTypeName*.java – The SEI

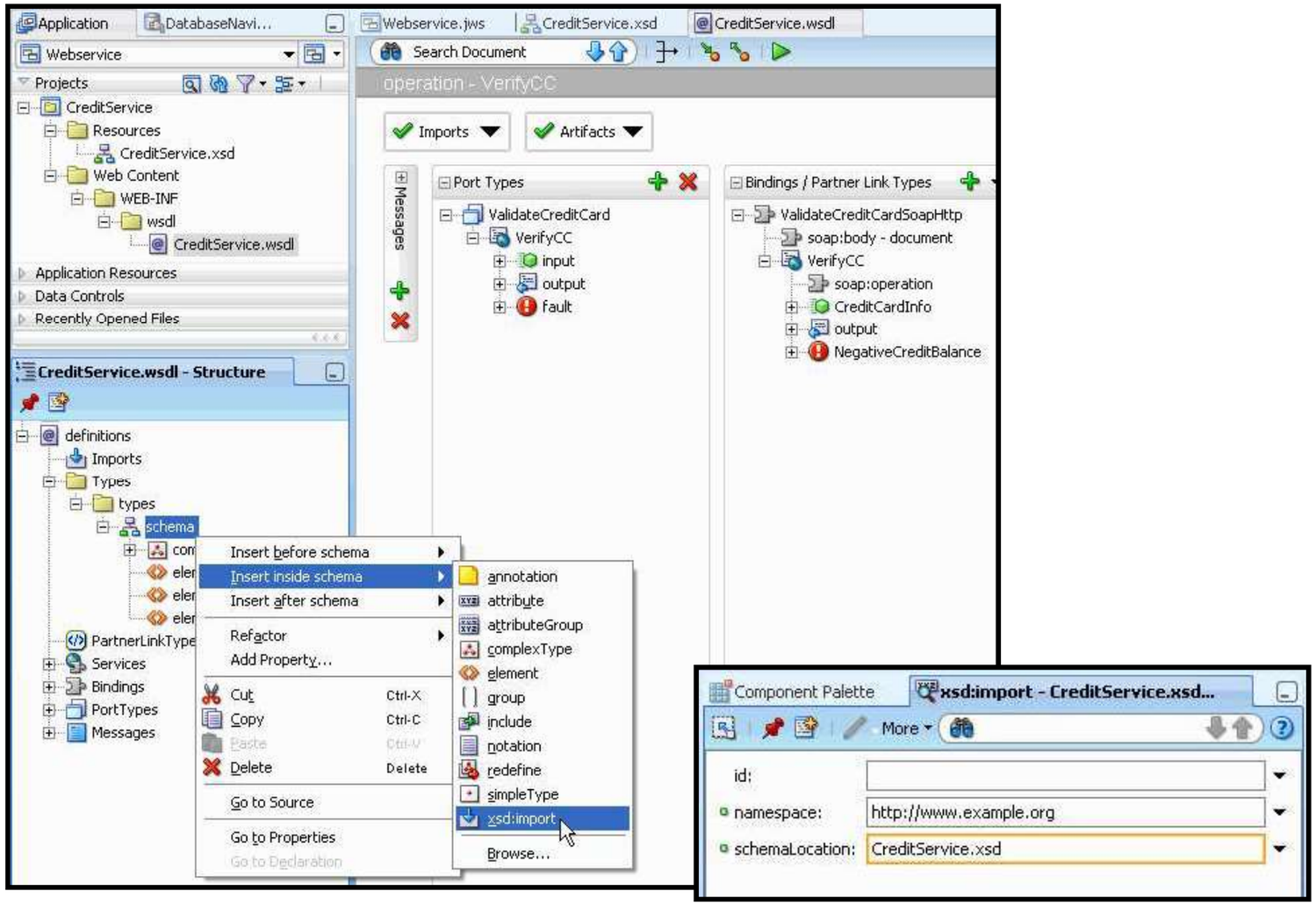
Generating Implementation Artifacts

The `wsimport` tool is a command-line tool to import a WSDL and to generate a SEI interface.

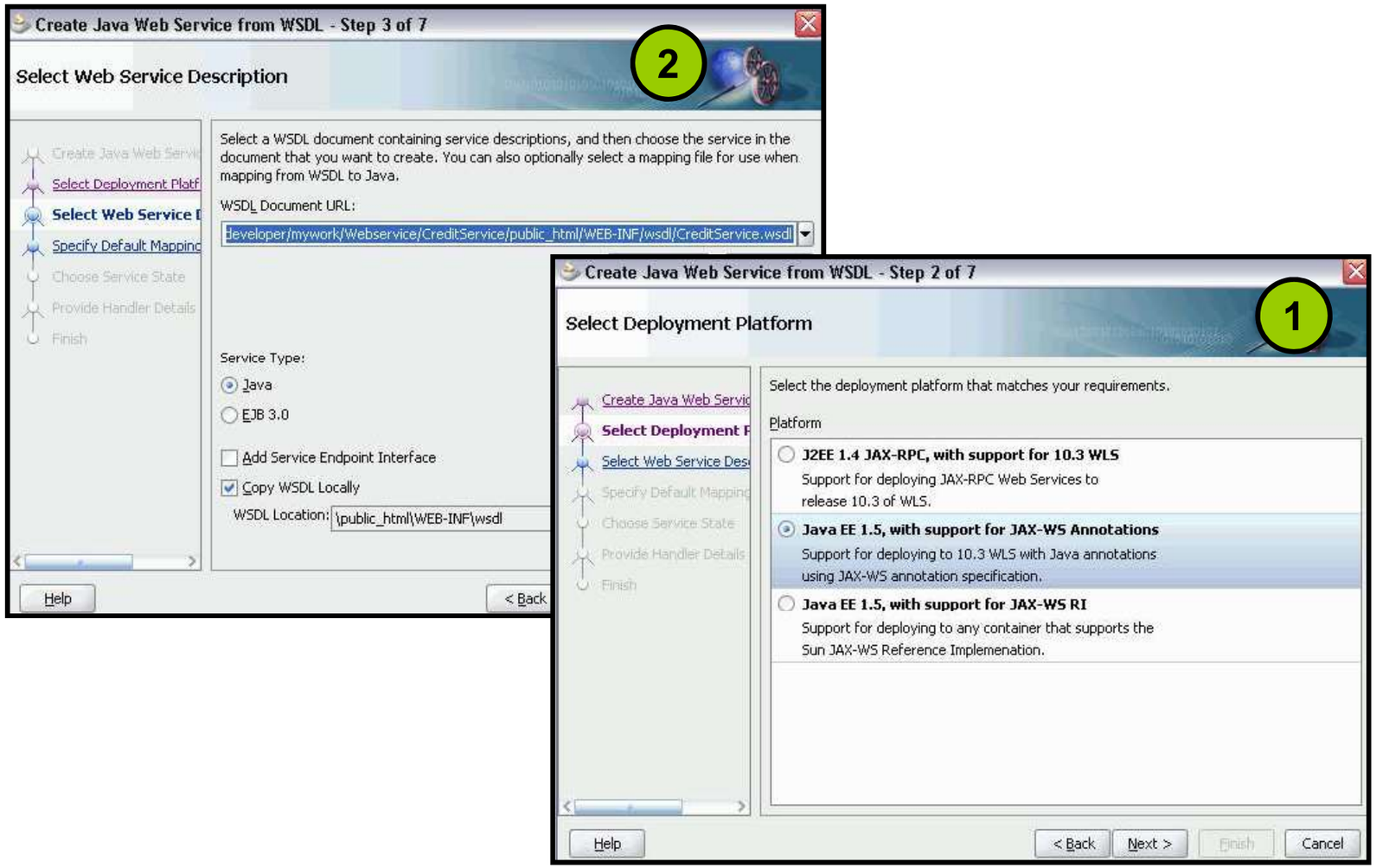
- Creating a "Web Service from WSDL" in NetBeans will run `wsimport`.
- The generated interface can be:
  - Implemented on the server to build a web service
  - Used on the client to invoke the web service
- Some of the important command-line switches include:
  - `-d` – location of generated class files
  - `-s` – location of generated source files

```
wsimport http://host:port/path?wsdl
wsimport PassengerManagerService.wsdl
```

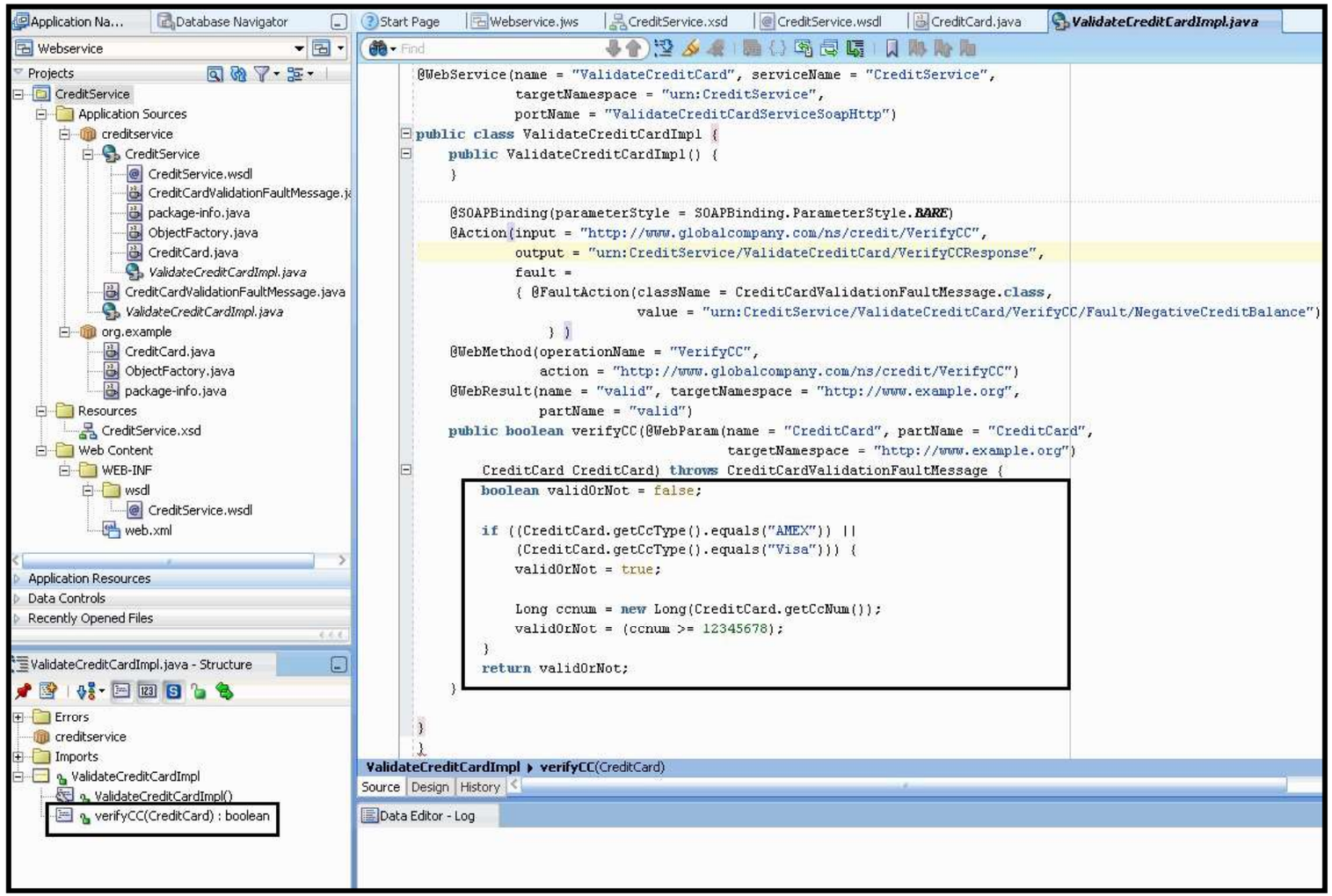
Modifying the WSDL Document



Creating the Web Service by Using



Implementing the Web Service Logic



Generated Java SEI

```
1 package com.example.generated;
2 @WebService(name = "PassengerManager",
3              targetNamespace = "urn://Traveller/")
4 public interface PassengerManager {
5    @WebMethod
6    public long addPassenger(
7        @WebParam(name = "firstName",
8                  targetNamespace = "urn://Traveller/")
9        String firstName,
10       @WebParam(name = "lastName",
11                 targetNamespace = "urn://Traveller/")
12       String lastName);
13 }
```

Service Implementation Class

```
1  @WebService(
2    endpointInterface=
3    "com.example.generated.PassengerManager")
4  public class PassengerManager
5  implements com.example.generated.PassengerManager {
6   public long
7   addPassenger(String firstName, String lastName) {
8     Passenger newPassenger =
9       new Passenger(firstName, lastName, null, null);
10    return dao.add(null, newPassenger ).getId();
11  }
12  private PassengerDAO dao = new PassengerDAO();
13 }
```

Customizing Generated Code

- Just as in the Java-to-WSDL scenario, one can customize:
  - The package name where generated classes will be placed
  - The name for the type generated for a WS port
  - The name for the method generated for a WS operation
    - Developer can introduce overloading in the generated code
- There are two ways to specify this customization detail:
  - Embedded declarations within the WSDL file
  - Separate customization XML file

Embedded Customization:
Custom Package Name

```
1  <definitions name="CustomPassengerManagerService.wsdl"
2      xmlns="http://schemas.xmlsoap.org/wsdl/">
3    <jaxws:bindings
4      xmlns:jaxws="http://java.sun.com/xml/ns/jaxws">
5      <jaxws:package name="com.example.custom"/>
6  </jaxws:bindings>
7  <service name="CustomPassengerManagerService">
8    <port name="CustomPassengerManager"
9       binding="tns:CustomPassengerManagerBinding">
10     <soap:address
11       location="http://localhost:8080/customManager"/>
12   </port>
13 </service>
```

Embedded Customization:
Custom Class Name

```
1  <definitions name="CustomPassengerManagerPort.wsdl"
2     xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
3     xmlns="http://schemas.xmlsoap.org/wsdl/">
4    <!-- types -->
5    <portType name="CustomPassengerManager">
6      <jaxws:bindings>
7        <jaxws:class name="CustomManager"/>
8      </jaxws:bindings>
9      <!-- operations ... -->
10   </portType>
11 </definitions>
```

Embedded Customization:
Custom Method Name

```
1  <definitions name="CustomPassengerManagerPort.wsdl"
2    xmlns:jaxws="http://java.sun.com/xml/ns/jaxws"
3    xmlns="http://schemas.xmlsoap.org/wsdl/">
4   <!-- types -->
5   <portType name="CustomPassengerManager">
6      <operation name="addPassenger">
7        <jaxws:bindings>
8          <jaxws:method name="add"/>
9          </jaxws:bindings>
10       <!-- messages ... -->
11     </operation>
12  </portType>
13 </definitions>
```

Generated Code:
From Customized WSDL

```
1  package com.example.custom;
2  @WebService(name = "CustomPassengerManager",
3              targetNamespace = "urn://Traveller")
4  public interface CustomManager {
5     @WebMethod(operationName = "addPassenger")
6     public long add(
7        @WebParam(name = "firstName",
8                  targetNamespace = "urn://Traveller")
9        String firstName,
10       @WebParam(name = "lastName",
11                 targetNamespace = "urn://Traveller")
12       String lastName);
13 }
```

JAX-WS will not validate web service calls against XML schema constraints unless requested:

```
1  @SchemaValidation
2  @WebService(
3    endpointInterface=
4    "com.example.generated.PassengerManager")
5  public class PassengerManager
6  implements com.example.generated.PassengerManager {
7    public long
8    addPassenger(String firstName, String lastName) {
9     Passenger newPassenger =
10      new Passenger(firstName, lastName, null, null);
11    return dao.add(null, newPassenger ).getId();
12   }
```

Comparing Development Approaches

- Each of the development approaches has benefits and costs, and is best suited to specific scenarios.
- You should decide on a specific development approach based on the needs of the development effort.

When using a top-down development approach to create SOAP endpoints the resulting method names are determined solely by the WSDI file.

a. True

b. False

To generate the service endpoint interface from a WSDL you would use:

a. apt
b. javac
c. wsgen
d. wsimport

Resources

Topic	Website
Getting Started With JAX-WS Web Services for Oracle WebLogic Server	http://docs.oracle.com/cd/E24329_01/web.1211/e24964/toc.htm
Programming Advanced Features of JAX-WS Web Services for Oracle WebLogic Server	http://docs.oracle.com/cd/E24329_01/web.1211/e24965/toc.htm
The Java EE 6 Tutorial - Building Web Services with JAX-WS	http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html

Summary

In this lesson, you should have learned how to:

- Describe the benefits of WSDL First Design
- Generate Service Endpoint Interfaces (SEIs) from WSDLs
- Implement Service Endpoint Interfaces
- Customize SEI Generation

Practice 9 : Overview

This practice covers the following topics:

- Creating the Player Management Servicer