8

Bottom-Up JAX-WS Web Services

Objectives

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

- Describe the benefits of Code First Design
- Create JAX-WS POJO Endpoints
- Create JAX-WS EJB Endpoints

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** — **You are here!**
- Lesson 9: Top Down JAX Web Services
- Lesson 10: Implementing JAX RS Web Services

Course Roadmap

Application Development Using Webservices [SOAP and Restful]

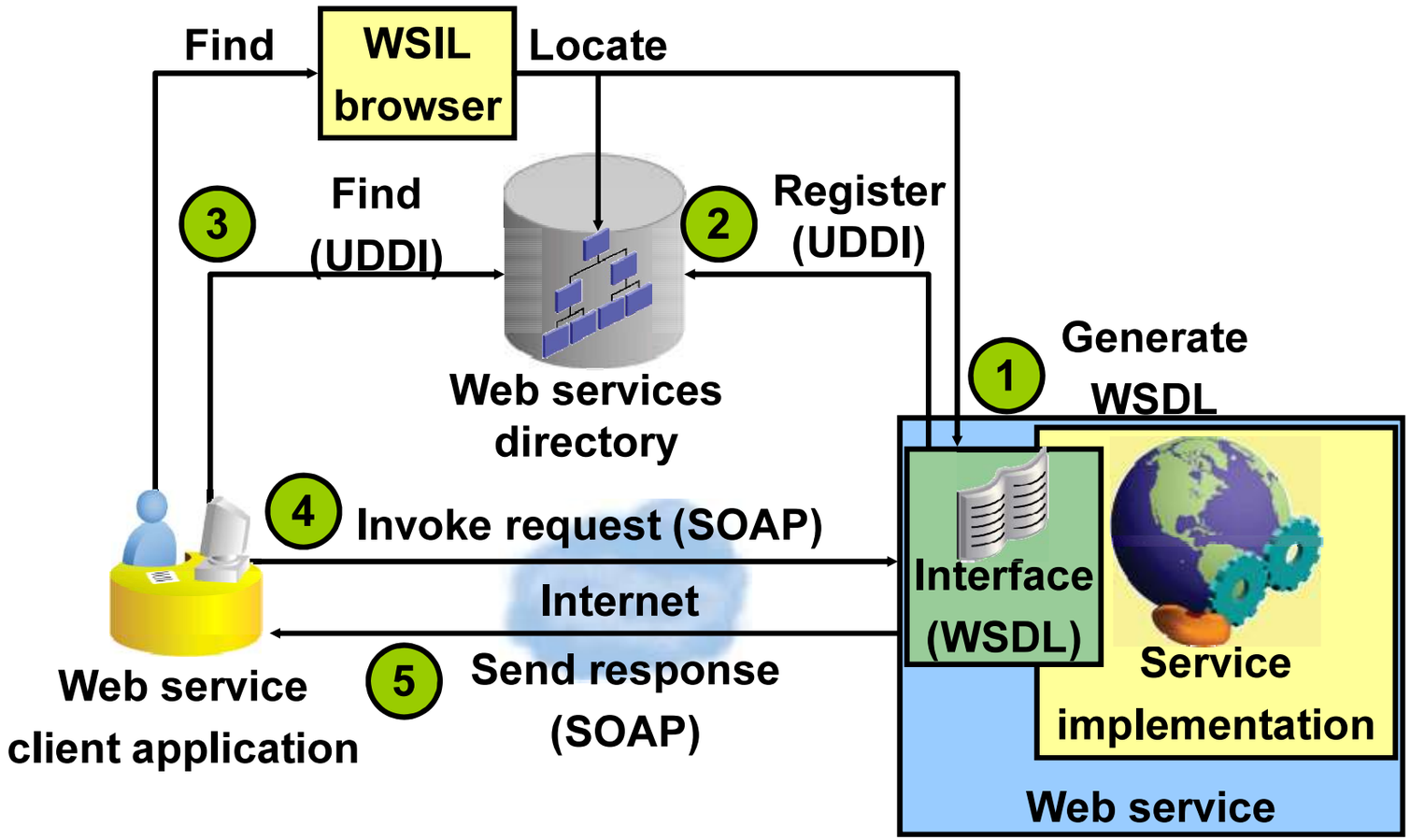
- Lesson 11: Web Service Error Handling
- Lesson 12: Java EE Security and Securing JAX WS

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.

Web Service Architecture

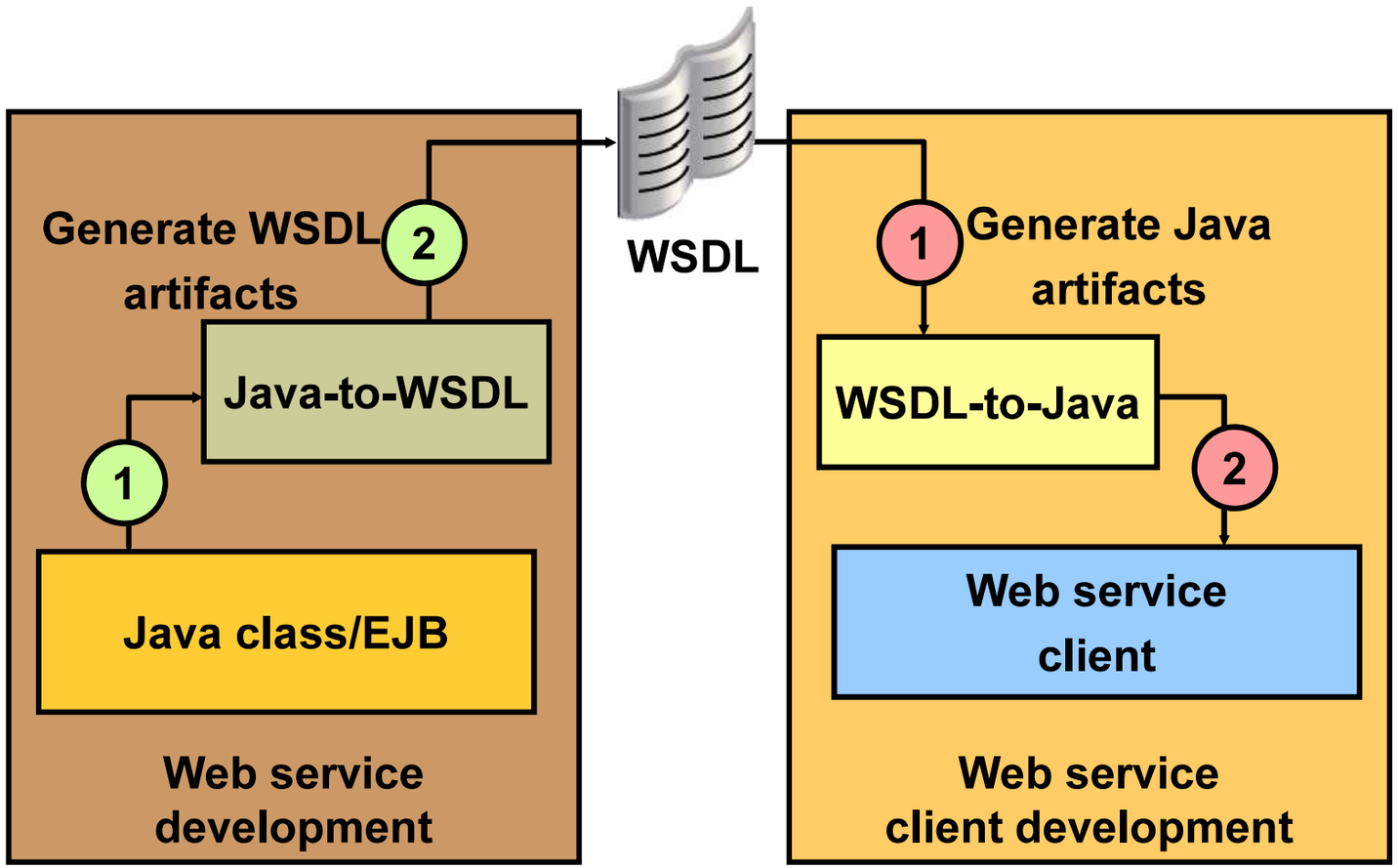


Benefits of a Bottom-Up Approach

When you provide a web service interface to an existing enterprise application, you should work from Java code:

- You can use the quickest development path.
- It is a natural approach, especially when business logic has already been implemented.
- You can map existing domain models directly to WSDL with little effort.
- You can re-use a service facade as a mediator to the domain logic for other types of applications and clients.

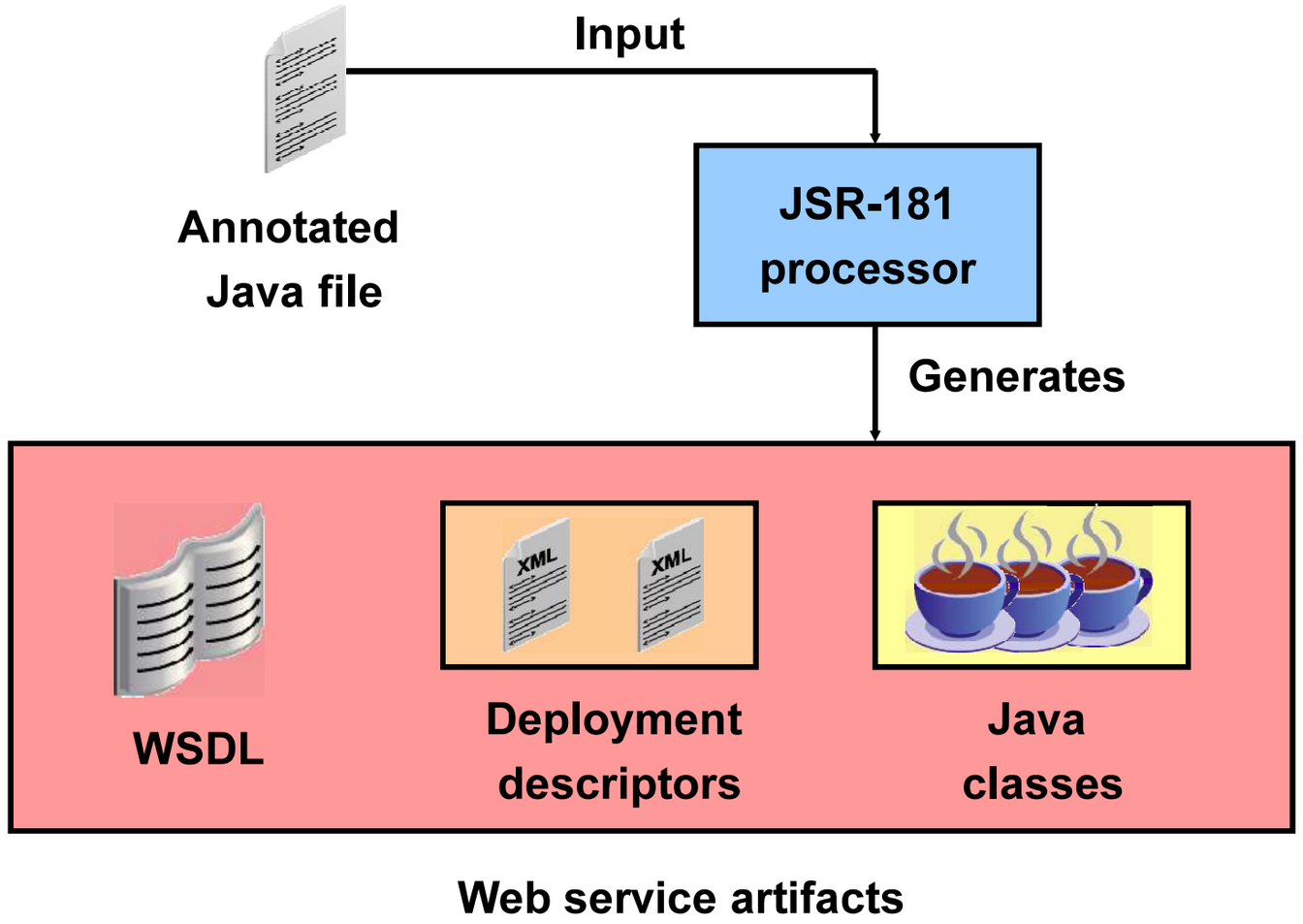
Bottom-Up Approach



Web Service Metadata for

- Is a Java specification to develop Web services by using a document written in annotated Java syntax
- Defines a processor model that acts as a bridge between the annotated document and the Web service artifacts

Examining the JSR-181 Processor Model

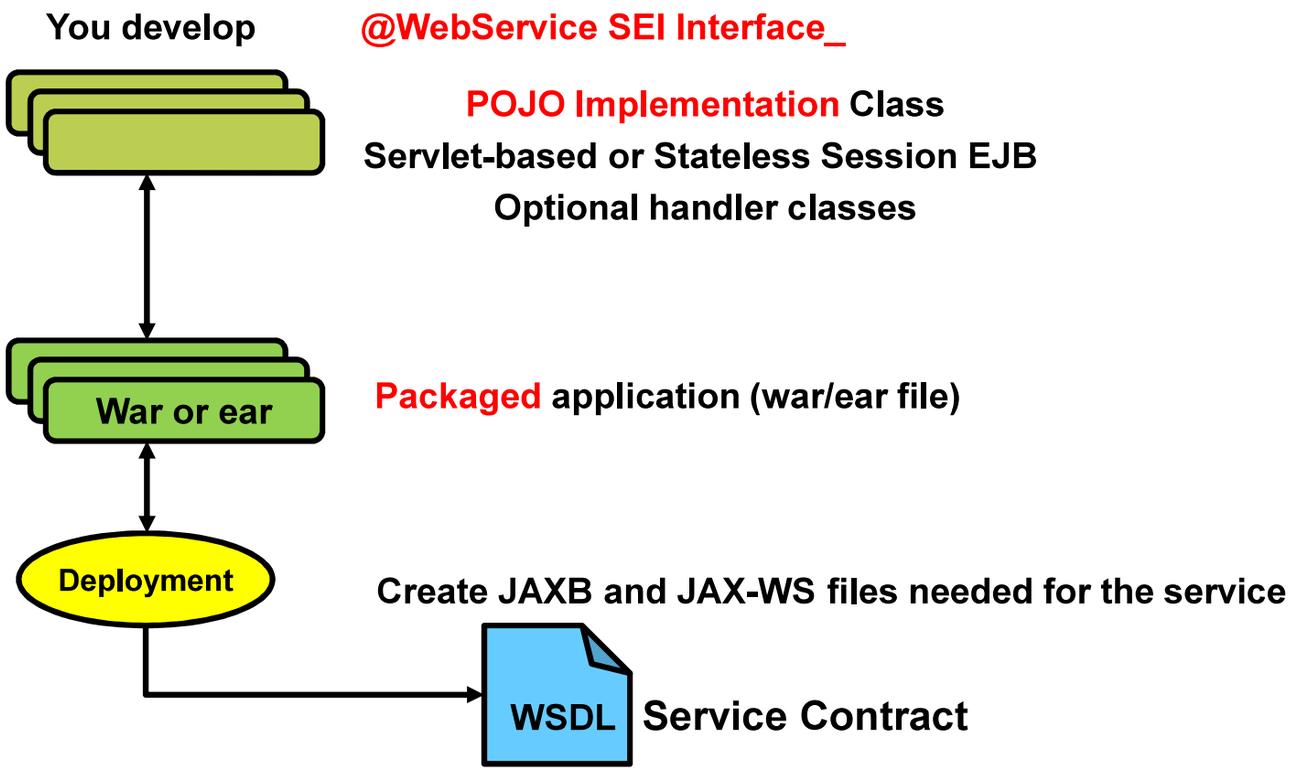


Describing JSR-181 Annotations

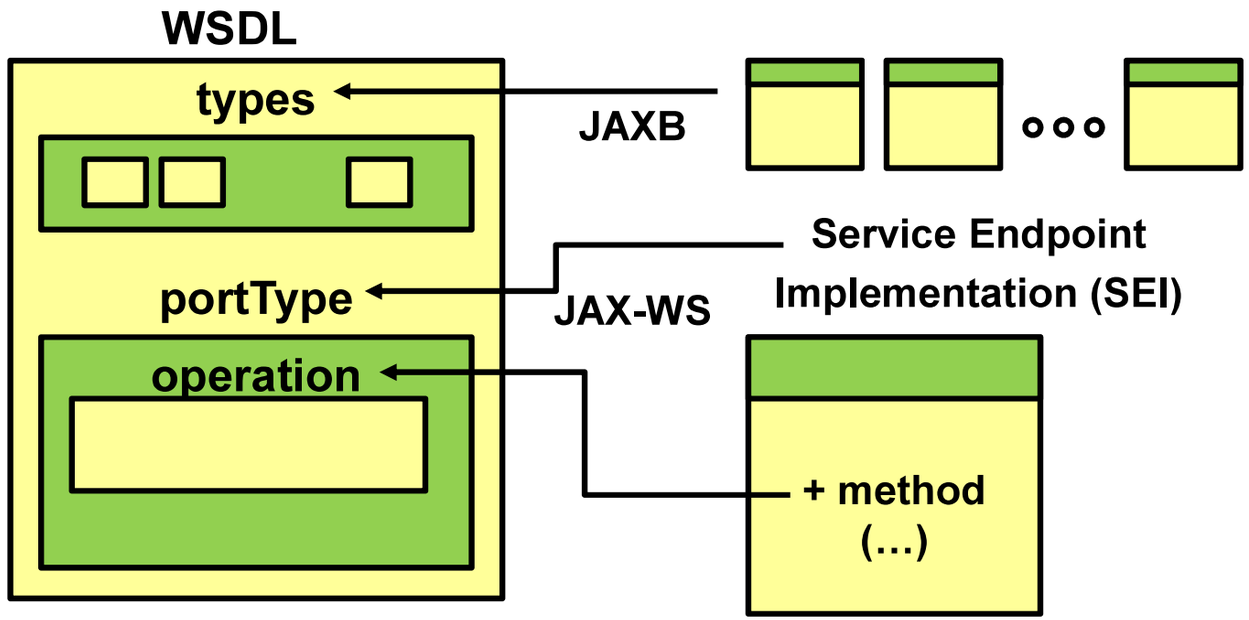
Standard annotations defined by JSR-181:

Annotation	Java Class or Interface
`@WebService`	`javax.jws.WebService`
`@WebMethod`	`javax.jws.WebMethod`
`@WebParam`	`javax.jws.WebParam`
`@WebResult`	`javax.jws.WebResult`
`@SOAPBinding`	`javax.jws.soap.SOAPBinding`

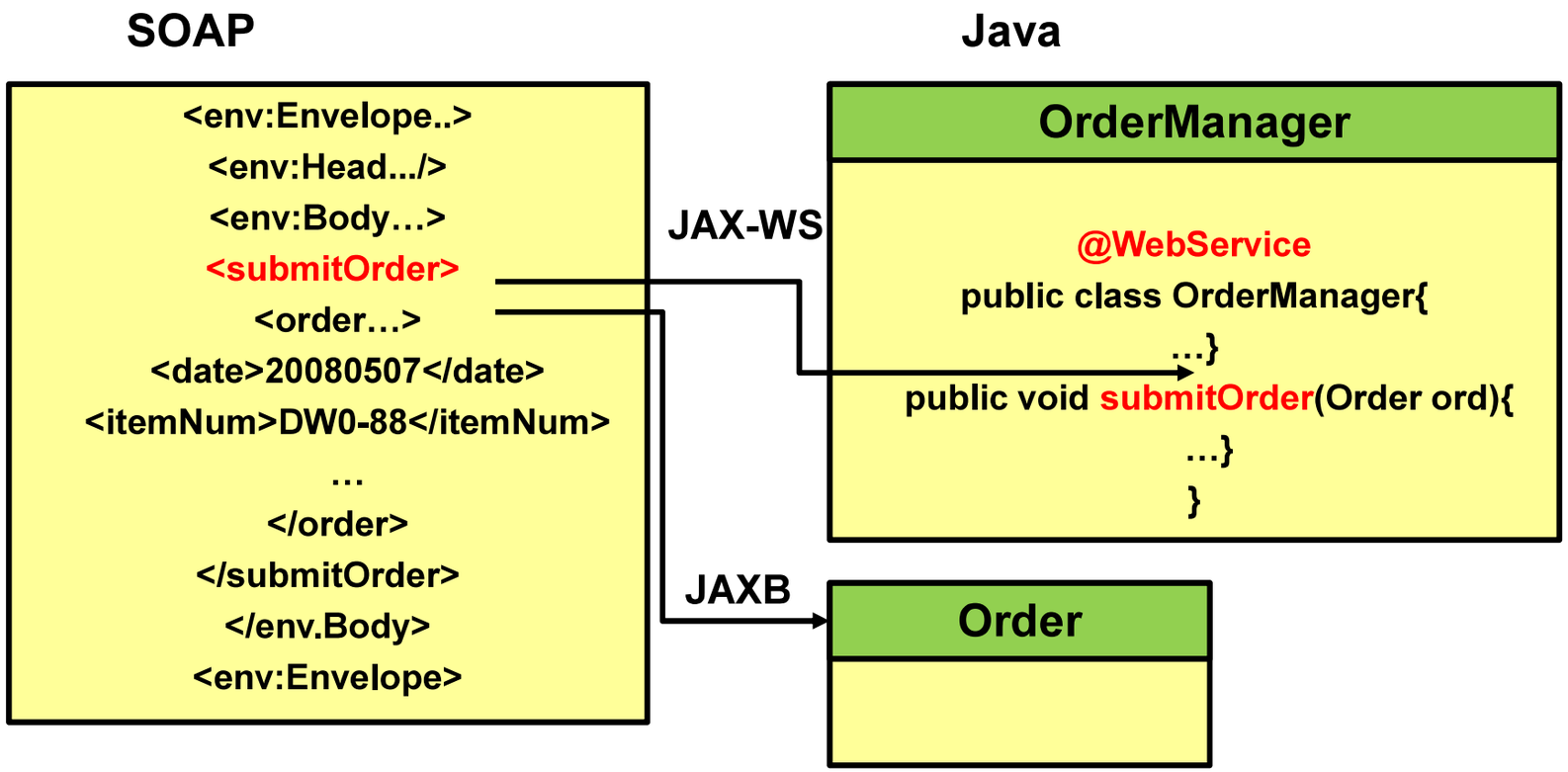
Starting from a Java Class



Java to WSDL


WSDL
types
portType
operation
JAXB
Service Endpoint
Implementation (SEI)
JAX-WS
+ method
(…)

JAX-WS:
SOAP to Java



A Simple Service: AirportManager

com.example.standalone.AirportManager

```
1  public class AirportManager {
2    public long
3    addAirport(String code, String name) {
4      return dao.add(null, code, name).getId();
5    }
6    private AirportDAO dao = new AirportDAO();
7  }
```

Simplest Web Service

com.example.jaxws.server.AirportManager

```
1  @WebService
2  public class AirportManager {
3    public long
4    addAirport(String code, String name) {
5      return dao.add(null, code, name).getId();
6    }
7    private AirportDAO dao = new AirportDAO();
8  }
```

Life Cycle

- A Service Implementation must be a stateless object.
- A Service Implementation Bean must not save a client-specific state across method calls either within the bean instance's data members or outside the instance.
- A container may use any bean instance to service a request including:
  - A new bean instance per request
  - The same bean instance used across requests
- The `@PostConstruct` and `@PreDestory` annotations can be used for life cycle methods.

JAX-WS Requirements:

Requirements on the Java Class

- Must be annotated with `javax.ws.WebService`
- Must not be declared `final`
- Must not be `abstract`
- Must have a default public no-arg constructor
- Must not have a `finalize` method

Web Service URLs

The default URL for a web service is based on the classname of the service.

- POJO endpoints – The default URL is:

  `http://host:port/{app}/{classname}Service`

  Use of `@WebService(serviceName="mypath")` results in:

  `http://host:port/{app}/mypath`

- EJB endpoints – The default URL is:

  `http://host:port/{classname}/{classname}Service`

  Use of `@WebService(serviceName="mypath")` results in:

  `http://host:port/{classname}/mypath`

JAX-WS Requirements:

Requirements on Web Service Methods

- Must be `public`

  By default, every public method in the class will be part of the web service.
- Methods can be excluded as web service methods using `@WebMethod(exclude=true)`.
- 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.

Generating the WSDL:

WSDL Description and Supporting Files

To publish a SEI there must be a WSDL per service and JAXB classes for each SOAP message.

- JAX-WS can deliver WSDL and message classes dynamically. The WSDL is available at:

  `http://host:port/path/to/service?WSDL`

Obtaining the WSDL from a running service is preferred. If you need to generate a WSDL without running the SEI:

- Use the annotation processor in `jaxws-tools.jar`

```
javac  -processor com.sun.tools.ws.processor.modeler.annotation.WebServiceAp
```

- Use `apt` or `wsgen` (deprecated in Java 7)

```
apt [-d outputDir] sourceFile ...
wsgen [-d outputDir] classFile ...
```

Custom WSDL Description:

Overloaded Methods

Web services require unique names for each method:

```
1  @WebService
2  public class BetterAirportManager {
3   @WebMethod(operationName="removeById")
4   public void removeAirport(long id) {
5   }
6   @WebMethod(operationName="removeByCode")
7   public void removeAirport(String code) {
8   }
9   // ...
10 }
```

Generated WSDL:
"Overloaded" Operations

```
1  <portType name="BetterAirportManager">
2    <!-- ... -->
3    <operation name="removeById">
4      <input message="tns:removeById"/>
5      <output message="tns:removeByIdResponse"/>
6    </operation>
7    <operation name="removeByCode">
8      <input message="tns:removeByCode"/>
9      <output message="tns:removeByCodeResponse"/>
10   </operation>
11 </portType>
```

Custom WSDL Description:

Default XML Namespace

Default XML namespace is computed:

```
package com.example.security.server;
```

produces

```
1  <definitions
2    targetNamespace=
3      "http://server.security.example.com/"
4    ... />
```

Custom WSDL Description:

Custom XML Namespace

Set `targetNamespace` attribute to `@WebService`

```
1  @WebService(
2    targetNamespace="urn://com.example.managerNS")
3  public class NamespacedAirportManager {
```

produces

```
1  <definitions
2    targetNamespace="urn://com.example.managerNS/"
3    ... />
```

Custom WSDL Description:

Custom Operation Parameter Names

The default names of method parameters are arg0, arg1, and so on:

```
<xs:complexType name="getMessage">
<xs:sequence>
<xs:element name="arg0" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
```

To modify the element names used for method parameters, use the `@WebParam` annotation.

```
public String getMessage(@WebParam(name="message") String
message) {
    String s = this.getClass().getName() + "'s count is " +
count;
    System.out.println(s);
    return s;
}
```

WebServiceContext

A WebServiceContext allows an endpoint to access message context and security information.

In the endpoint class:

```
@Resource
private WebServiceContext context;
```

In a method of the endpoint class:

```
for(String key : context.getMessageContext().keySet()) {
    Object obj = context.getMessageContext().get(key);
    if(obj != null) {
        System.out.println("Context=" + key + ":" +
                           obj.getClass() + "=" + obj);
    }
}
```

Simple Stand-Alone Server

```
1  public class AirportManager {
2   // ...
3   static public void main(String[] args) {
4    String url =
5      "http://localhost:8080/airportManager";
6    if (args.length > 0)
7     url = args[1];
8    AirportManager manager = new AirportManager();
9    Endpoint endpoint =
10     Endpoint.publish(url, manager);
11  }
12 }
```

Finer Control over Stand-Alone Server

```
1  public class FancyServer {
2   public static
3   void main( String[] args ) throws Exception {
4       HttpServer server =
5        HttpServer.create(new InetSocketAddress(8080),10);
6       Executor executor = Executors.newFixedThreadPool(10);
7       server.setExecutor(executor);
8       HttpContext context =
9         server.createContext("/fancyServer");
10      AirportManager manager = new AirportManager();
11      Endpoint endpoint = Endpoint.create(manager);
12      endpoint.publish(context);
13      server.start();
14   }
15 }
```

Viewing SOAP Messages

Sometimes, it helps to dump all SOAP messages exchanged by the server to standard out:

- JAX-WS includes a generic framework that can be used for this: JAX-WS Handlers.
- Use a system property to dump all JAX-WS messages.

```
-Dcom.sun.xml.ws.transport.http.HttpAdapter.dump=true
```

SOAP 1.2

You may use SOAP 1.2 by adding a `@BindingType` annotation to a service endpoint class.

```
@BindingType(value=SOAPBinding.SOAP12HTTP_BINDING)
```

The reference implementation does not currently support WSDL 2.0.

Testing a Web Service

- In WebLogic Server, `http://localhost:7001/wls_utc/`
  - Operations with complex types are supported.
- In Glassfish, `http://localhost:8080/webService?Tester`
  - Operations with complex types are not supported.
- Any HTTP tool that can submit a POST request with a custom body.
  - Firefox RESTClient
  - cURL
- JDeveloper's HTTP Analyzer
- Dedicated SOAP web service testing applications
  - SoapUI

Public methods in an endpoint must be annotated with @WebMethod in order to be exposed to clients.

a. True
b. False

When using a bottom-up development approach to create SOAP endpoints you create the WSDL file first.

a. True

b. False

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 Code First Design
- Create JAX-WS POJO Endpoints
- Create JAX-WS EJB Endpoints

Practice 8 : Overview

This practice covers the following topics:

- Creating the Card Game Service
- Publishing Endpoints Without an Application Server