INTRODUCTION TO SOAP WS

Web Services

- A client and server applications that communicate over HTTP or HTTPS
  - A standard means of **interoperating between software applications** running on a *variety of platforms and frameworks*.
- Characterized by machine-processable descriptions, with the use of XML.
  - Can be combined in a loosely coupled way to achieve complex operations.

Types of Web Services

- *A service is a software component provided through a network-accessible endpoint.*

- The service consumer and provider use messages to exchange request and response
  - Done by a self-containing documents with very few assumptions about the technological capabilities of the receiver.

- Can be implemented as
- **“big” web services and “RESTful” web services.**

“Big” Web Services

- Also known as SOAP Web Services
- Uses XML messages that follow the Simple Object Access Protocol
- JAX-WS provides the functionality

- Contains a machine-readable description of the operations offered by the service
  - Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

XML MESSAGING-SOAP

SOAP- Definition

- Simple Object Access Protocol
  - Supports XML-based RPC (Remote Procedure Call)
- Its a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment
- Uses XML technologies to define an extensible messaging framework providing a message construct
- Can be used in combination with HTTP and HTTPS
- The framework has been designed to be independent of any particular programming model and other implementation specific semantics

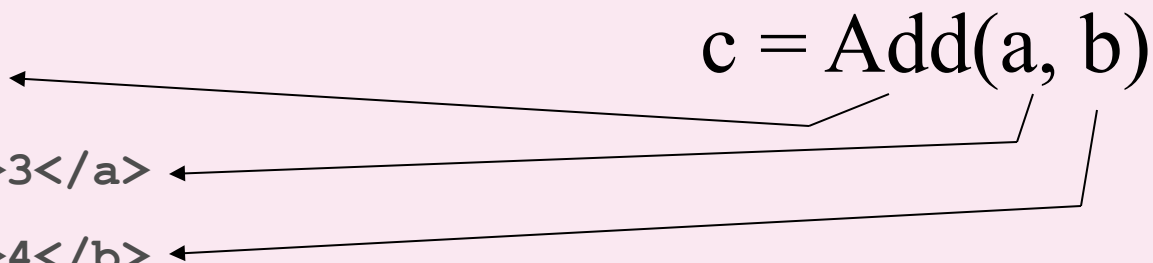
Parts of SOAP

- The Three Parts of SOAP
  - an **envelope** that defines a framework for describing what is in a message and how to process it,
  - a set of **encoding rules** for expressing instances of application-defined datatypes,
  - a convention for representing **remote procedure calls** and responses.
- Both One way messaging and Request Response Messaging messages travel as SOAP message

Simple Example

```
<Envelope>
    <Header>
        <transId>1234</transId>
    </Header>
    <Body>
        <Add>
            <a>3</a>
            <b>4</b>
        </Add>
    </Body>

</Envelope>
```



System Flow (HTTP)

```
<Envelope>
   <Header>
      <transId>1234</transId>
   </Header>
   <Body>
      <Add>
         <a>3</a>
         <b>4</b>
      </Add>
   </Body>
</Envelope>
```

Server

Request

Response

Client

```
<Envelope>
   <Header>
      <transId>1234</transId>
   </Header>
   <Body>
      <AddResponse>
         <c>7</c>
      </AddResponse>
   </Body>
</Envelope>
```

Actual SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">1234</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m="http://a.com/Calculator">
         <a xsi:type="integer">3</a>
         <b xsi:type="integer">4</b>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Actual SOAP Response

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">1234</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m="http://a.com/Calculator">
         <c xsi:type="integer">7</c>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

JAX-WS

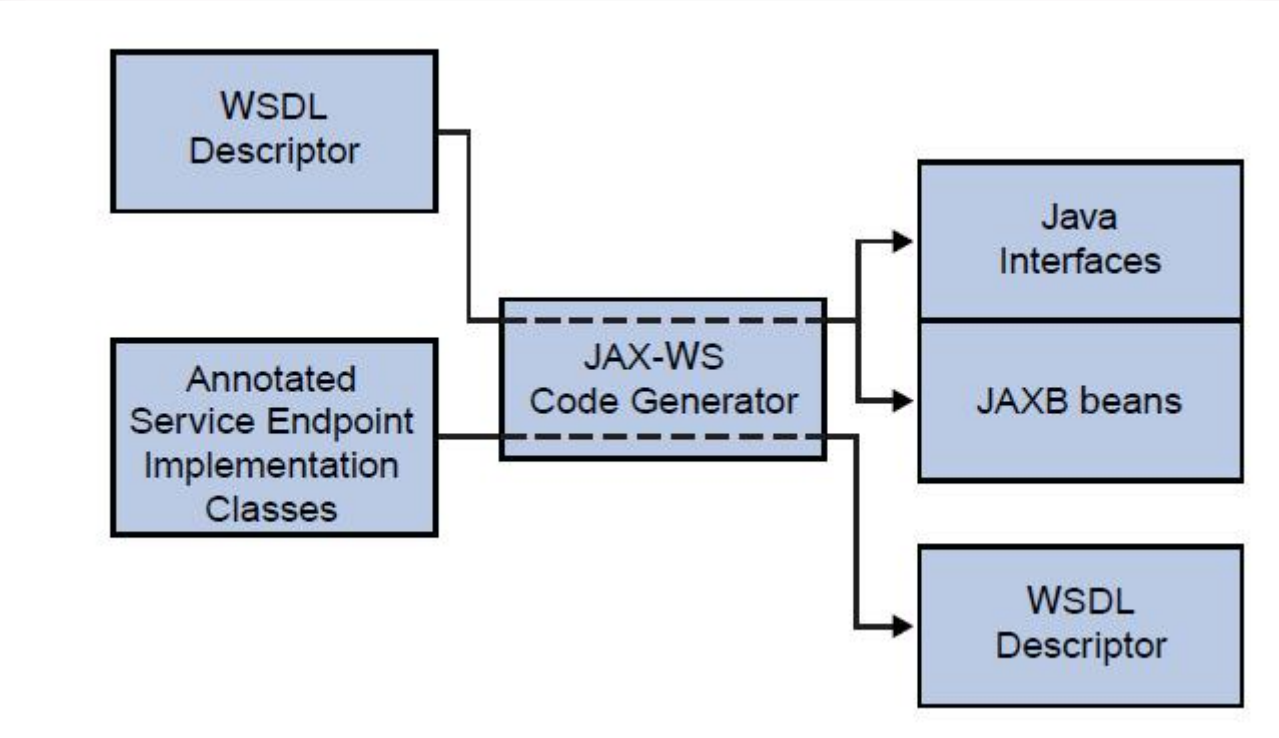
Developing a Web service

- Contract-first approach versus code-first approach
- **Contract first**
  - **Start with a WSDL contract, and generate a Java class to** implement the service.
  - Requires a good understanding of WSDL and XSD
- **Code first**
  - **Start with a Java class, and use annotations to generate both** a WSDL file and a Java interface.

Bottom-up approach – Start with Java Classes

- Service End Point interface is created first
  - SEI typically is a standard Java interface.
  - Will become the "*wsdl:portType* " element WSDL document.
- Methods are defined in SEI to expose as services.
  - It will become the "***wsdl:operation***" elements in the "***wsdl:portType***"

JAX-WS Artificats



Service EndPoint interfaces

- Java interface that declares the method that the client can invoke on the service.
- The implementation class implicitly defines a SEI if it is not specified in the implementation class.
- Can specify an explicit SEI by adding the endpointinterface element to the WebService annotation in the implementation class.

WebService Class Requirement

- An implementation must support Java annotations.
- The implementing class be **annotated with @WebService**
- Must **NOT** be declared **final** and must **NOT** be **abstract**.
- Must have a **default public constructor** and **Must NOT define** the **finalize** method.

@Web Service

- **WebService：Marked in endpoint implementation class or service endpoint interface**
- Name:
  - Specifies the name of the service interface
  - Mapped to the name attribute of the wsdl:portType in WSDL
- targetNamespace
  - Specifies the target namespace under which the service is defined.
  - Default value is package name.
- serviceName
  - Specifies the name of the published service.
  - Mapped to the name attribute of the wsdl:service in WSDL
  - Default value is name of SEI implementation class.
  - This attribute is used in implementation class.

@Web Service

- **wsdlLocation**
  - Specifies the URI at which the service's WSDL contract is stored.
  - Default value is the URI at which the service is deployed
  - used when using the meet-in-the-middle approach
- **endpointInterface**
  - Qualified name of the service endpoint interface
  - Specifies the full name of the SEI that the implementation class implements.
  - This attribute is used in implementation class.
- **portName**
  - The name of the endpoint at which the service is published
  - Mapped to the name attribute of the wsdl:port in WSDL
  - Default value is to append Port to the name of the service's implementation class.
  - This is used in implementation class.

@WebMethod

- Marked on Pojo Class Methods
- Information that is normally represented as wsdl:operation in WSDL.
- This indicates the operation to which the method is associated.

- **operationName** :
  - Defaults to the name of the Java method.
  - Indicates the value of the associated wsdl:operation element's name..

- **exclude :**
  - Marks a method NOT to be exposed as a web method.
  - Used to stop an inherited method from being exposed as part of this web service.
  - Default false.

The @WebParam annotation

- Placed on the parameters on the methods defined in the SEI.
- Parameter will be placed in the SOAP header, and other properties of the generated *wsdl:part*.
- Has the following attributes.

- name
  - Specifies the name of the parameter as it appears in the WSDL.
  - **For RPC bindings**, this is **name of the *wsdl:part*** representing the parameter.
  - For **document bindings,** this is the **local name of the XML element** representing the parameter.
- targetNamespace:
  - Indicates the namespace for the parameter
  - Defaults is to use the service's namespace.

@WebResult

- To specify the properties of the generated *wsdl:part* that is generated for the method's return value.
- This annotation is placed on the methods defined in the SEI.
- This annotations has *name*, *targetNamespace*, header and *partName* attributes in which header specifies if the return value is passed as part of the SOAP header.

JAX-WS- Web Service

```
package com.training.ifaces;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding(style=Style.RPC)

public interface CurrencyConverter {

    public double dollorToRupees(double dlrAmt);
}
```

Business Method

- Creating aWeb Service Using JAX-WS
- The business methods of the implementing class that are exposed to web service clients:
- Must be public and must not be static or final.
- Can be annotated with javax.jws.WebMethod; however, it is not mandatory.
- Must have JAXB-compatible parameters and return types.
- A method parameter or return type must not implement the java.rmi.Remote interface either directly or indirectly.

Implementation

```
@WebService(endpointInterface="com.training.ifaces.CurrencyConverter")

public class CurrencyConverterImpl  implements CurrencyConverter{

    @Override
    public double dollorToRupees(double dlrAmt) {

        return dlrAmt * 52.00d;
    }


}
```

Publisher

```
public static void main(String[] args) {

    try {

        Endpoint.publish("http://127.0.0.1:9000/converter",
                                    new CurrencyConverterImpl());

        System.out.println("Service Published");

    } catch (Exception e) {
            e.printStackTrace();
             }
    }
```

Endpoint

- An endpoint can be created using any of the following constructors:

  Endpoint.create(implementor)

  Endpoint.**create**(bindingId,**implementor)**

- Endpoint can be published using the following method.

  – *Endpoint.publish(address, implementor)*

- Any published Endpoint can be stopped using Endpoint.stop()

Developing JAX-WS Clients

- A JAX-WS client can use the WSDL and a Proxy object to handle the creation of SOAP messages and HTTP communication.
- This proxy type class is know as a Port in JAX-WS.
- Required artifacts for the web service can be creatred by application known as wsimport
- The wsimport application can also be run as an Ant task
- Can Use AJAX-WS
- Clients need to have access to the WSDL file
- Support the correct version of the SOAP specification.

Utility Command - wsimport

- **The wsimport command-line** tool processes the W SDL file with schema definitions to generate the portable artifacts

- The wsimport tool generates JAX-W S portable artifacts

  - Service Endpoint Interface (SEI)
  - Service
  - Exception class mapped from wsdl:fault (if any)
  - Async Reponse Bean derived from response wsdl:message (if any)
  - JAXB generated value types (mapped Java classes from schema types)

Utility Command - wsimport

- A command line tool to create Static Clients.
- Import a WSDL and to generate an SEI (a javax.jws.WebService) interface, the generated interface can be either:
  - Implemented on the server side to build a web service
  - Can be used on the client side to invoke the web service
- *wsimport -p com.training –keep http://localhost:3030/JaxWebService/price?wsdl*
- *wsimport –p com.training –keep Travel.wsdl*

specifies the target package

- - d -> The output location of generated class files
- - s -> The output location of generated source files

Client

- Dynamic proxy client
  - Invokes a Web service based on a Service Endpoint Interface (SEI)

```
try {

URL url = new URL("http://127.0.0.1:9000/converter");

    QName name = new QName("http://services.training.com/",
                                        "CurrencyConverterService");

        Service service = Service.create(url, name);

        CurrencyConverter obj = service.getPort(CurrencyConverter.class);

        System.out.println(obj.dollorToRupees(100));
    } catch (Exception e) {
            e.printStackTrace();
    }
```

Client

```
public static void main(String[] args) {


 try {
      CurrencyConverterServiceProxy proxy = new
        CurrencyConverterServiceProxy();


      System.out.println(proxy.dollorToRupees(100);
}
} catch (RemoteException e) {
e.printStackTrace();
}
}
```